Paralelización de un algoritmo de búsqueda de patrones en imágenes basado en la distancia de Hausdorff

Daniel Ortiz Nieto

16 de octubre de 2012

Tutores:

Luis Miguel Sánchez García, Antonio Berlanga de Jesús

ARCOS, GIAA

Escuela Politécnica Superior

Universidad Carlos III de Madrid

Índice general

Resumen

Este trabajo discute diversas técnicas y tecnologías de paralelización de código, estudia diferentes estrategias de análisis de imágenes. Posteriormente propone un algoritmo para detección de formas en imágenes haciendo uso de la distancia de Hausdorff y presenta una implementación secuencial y una paralelización en OpenMP. Finalmente compara el rendimiento entre estos desarrollos y expone las conclusiones obtenidas.

Capítulo 1

Introducción

El capitulo contiene el una breve descripción de la estructura del documento, el glosario de términos que se utilizan a lo largo de este, una breve descripción de los objetivos del presente trabajo, es decir, el problema que se plantea y la solución propuesta y desarrollada. Finalmente se incluye una planificación con una estimación de las horas dedicadas suponiendo una retribución estándar en el mercado de trabajo actual.

1.1. Estructura del documento

Este documento se divide en cuatro secciones principales, que a su vez contienen diferentes subsecciones que abordan diferentes aspectos del estudio realizado:

- **Introducción:** En esta sección se presentan los objetivos del proyecto así cómo una breve planificación, los detalles de estructura del documento y un glosario de términos.
- **Estado Del Arte:** Presentación del estado actual en tecnologías y herramientas de paralelización más relevantes al problema abordado y en las técnicas y métodos del análisis de imágenes 2D que pueden ser buenas candidatas para ser la base del algoritmo.
- **Desarrollo:** Explicación y justificación de las herramientas elegidas y del algoritmo presentado, así como las diferentes implementaciones realizadas.
- **Resultados y conclusiones:** La última sección contiene los detalles sobre la experimentación realizada y propone una serie de conclusiones que pueden extraerse de los resultados obtenidos.

A continuación de lo anterior se incluyen unos apéndices con información adicional que se considera relevante pero no esencial para el seguimiento del trabajo realizado.

1.2. Glosario

API Del Inglés *Application Programming Interface* es el conjunto de funciones y procedimientos que se ofrecen en forma de biblioteca para ser utilizado por otro software.

Apple Empresa multinacional que diseña y produce equipos electrónicos y software.

ATI Empresa especializada en la fabricación y diseño de hardware para procesamiento gráfico.

C/C++ Lenguaje de programación de propósito general híbrido, ya que puede manejar objetos o programación funcional, que presenta un modelo de codificación flexible y de alto rendimiento

Cilk Lenguaje de programación de propósito general específicamente diseñado para la programación paralela multihilo.

Cluster Conjunto o conglomerado de ordenadores construidos con hardware común y que están ideados para comportarse como una única máquina.

CPU Del Inglés *Central Processing Unit* es el componente principal de los ordenadores y demás dispositivos programables. Su función es interpretar instrucciones y procesar los datos que conforman los programas.

CUDA Siglas del Inglés *Compute Unified Device Architecture* son un conjunto de herramientas y un compilador creados por la compañía nVidia que permiten programar las GPU de la misma empresa con una versión ampliada del lenguaje C.

Dato Representación simbólica, un atributo o una caracteristica de una entidad. Es la unidad mínima de información que puede manipular un ordenador.

Fork En Español bifurcación, es la duplicación de un programa en memoria desde dentro del mismo. Es una práctica común en programación para repartir tareas o paralelizar el trabajo.

GPC Del Inglés *Graphics Computing Cluster* es la nomenclatura en las tecnologías de CUDA para referirse a las unidades de procesamiento grafico.

GPU Acrónimo Inglés de *Graphics Processing Unit* hace referencia a un coprocesador especializado en procesamiento de gráficos y operaciones en coma flotante, de forma que elimina esa carga de trabajo de la CPU.

Grid En Español sistema distribuido, es una colección de ordenadores separados físicamente y conectados entre sí por una red de comunicaciones; cada máquina posee sus componentes de hardware y software independientes, pero el usuario percibe el conjunto como un solo sistema.

Hash Tipo de función, conocida en Castellano como función resumen, que mapea las entradas a un rango de salida finito.

Instrucción Conjunto de datos insertados en una secuencia estructurada o específica que el procesador interpreta y ejecuta. Es la unidad mínima que un procesador puede computar en cada ciclo de reloj.

Intel Multinacional especializada en la fabricación de CPUs y creación de lenguajes o APIs de alto rendimiento.

Join Operación asociada al *Fork* donde varios, dos o más programas previamente separados vuelven a unirse en uno único.

Khronos Consorcio industrial financiado por sus miembros enfocado a la creación de APIs estándares abiertas y libres de cargo que permiten la creación y reproducción multimedia en un amplio abanico de plataformas y dispositivos.

Manycore Procesador multicore con una agregación de núcleos muy superior a la usual en los procesadores multinúcleo.

Memoria Dispositivos que retienen datos informáticos durante algún intervalo de tiempo.

MPI Estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

Multicore Procesador que combina dos o más nucleos de ejecución independientes en un solo circuito integrado.

nVidia Empresa Coreana especializada en la fabricación y diseño de GPUs y clusters de procesamiento de alto rendimiento con a su tecnología CUDA.

OpenCV Biblioteca libre de visión artificial originalmente desarrollada por la compañía Intel.

OpenGL Especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

OpenMP/OMP API diseñada para la programación multiproceso de memoria compartida en múltiples plataformas.

Open-Source Codigo abierto en castellano, es una filosofía de desarrollo de software que se fundamenta en la libre creación y distribución de éste. No confundir con el termino software libre, ya que aunque similares, no enfocan el mismo concepto.

PCIe Es un nuevo desarrollo del bus PCI que usa los conceptos de programación y los estándares de comunicación existentes, pero se basa en un sistema de comunicación serie mucho más rápido.

píxel Menor unidad homogénea en color que forma parte de una imagen digital.

Proceso Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.

SIMD Siglas del inglés *Single Instruction, Multiple Data* es una técnica empleada para conseguir paralelismo a nivel de datos.

SMX Unidad de procesamiento utilizado por nVidia en sus GPU.

Socket (de CPU) Zocalo en Castellano, es un sistema electromecánico de soporte y conexión eléctrica, instalado en la placa base, que se usa para fijar y conectar un microprocesador.

Tarea Ver proceso.

TBB Tecnología creada por Intel que corresponde con el acronimo *Threading Building Blocks* es una biblioteca basada en plantillas para C++ para facilitar la escritura de programas que exploten las capacidades de paralelismo de los procesadores con arquitectura multinúcleo.

Thread En Castellano hilo, es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

1.3. Objetivos

1.3.1. Problemática actual

El análisis y reconocimiento de imágenes ha sido uno de los principales focos de estudio y desarrollo desde la aparición de los primeros ordenadores. Usado en todo tipo de áreas desde el reconocimiento de patrones en imágenes de colisiones en física de altas energías, pasando por el estudio de mapas, la predicción del clima, hasta el reconocimiento facial y otras muchas. La necesidad de tener herramientas que faciliten el análisis y proporcionen unos resultados de calidad (entendida como una tasa baja de falsos positivos y negativos, resistencia al ruido y otra propiedades que se consideran en detalle en secciones posteriores), ha sido un problema de primera línea. Esta situación ha propiciado la creación de muchos métodos que permiten el reconocimiento, manipulación y análisis de imágenes o de formas y figuras dentro de imágenes, basados en sus propiedades físicas y matemáticas. En el la sección del Estado del Arte se consideran algunos de ellos en profundidad.

Esta diversidad de planteamientos teóricos sobre el análisis de imágenes ha dado lugar a una enorme competitividad entre herramientas, entre las que cada vez resulta más difícil decantarse por una en concreto para resolver un problema concreto. Las principales características que se recomienda considerar para decidir entre todas son las dos siguientes: en primer lugar la precisión y calidad del análisis y segundo el tiempo que éste tarda en generarse. El primer aspecto, la precisión y calidad, dependen de diversos factores, desde el objeto del análisis hasta ciertas restricciones teóricas que imponen algunos algoritmos. Desafortunadamente, en muchos métodos resulta muy difícil poder superar un determinado límite de imprecisión. La segunda cuestión, el tiempo, también está sujeto a determinados límites teóricos Amdahl (1967) y, por supuesto, de la máquina en donde se quieran realizar los estudios. Sin embargo, algunos algoritmos disponen de una ventaja sobre otros, y es que tienen la propiedad de ser paralelizables.

La paralelización es la acción de dividir una tarea y realizarla simultáneamente para resolverla en menos tiempo. Históricamente, la paralelización solo ha sido una práctica en grandes sistemas de computación, donde el volumen de trabajo requerido no puede afrontarse con una solución secuencial en un tiempo razonable. Con el alcance del límite al que se puede poner, de forma estable, la velocidad de los ciclos de un procesador, la paralelización se esta convirtiendo en una necesidad de todo tipo de aplicaciones informáticas. Esta solución ha hecho que surjan diferentes aproximaciones a la programación paralela, muchos de ellos impulsados por empresas que ofrecen sus productos de optimización y mejora de rendimiento. Se presentan y estudian las más importantes en la sección correspondiente del Estado del Arte.

Se encuentra por tanto que, con la extensión de las tecnologías paralelas, existe una gran oportunidad de destacarse en el campo del análisis de imágenes ofreciendo algoritmos de calidad que puedan aprovechar las ventajas de una implementación paralela.

1.3.2. Detalle de los Objetivos

En el presente trabajo se aborda el problema del reconocimiento de imágenes o de elementos en una imagen de forma eficiente. Para solucionar esta cuestión se plantean los siguientes objetivos:

1. Encontrar un concepto o propiedad matemática que permita el reconocimiento de imágenes que, además de cumplir con las condiciones deseables de robustez a la hora del análisis, posea una estructura de operación que facilite su optimización mediante la aplicación de técnicas de paralelización de las tareas necesarias.

2. Estudiar y seleccionar las mejores tecnologías de programación que nos permitan desarrollar una herramienta aplicando la idea anterior de la forma más optima posible.

3. Idear un algoritmo con este concepto y codificarlo con las tecnologías seleccionadas tanto de forma secuencial como paralela.

4. Generar un plan de pruebas y comparar los resultados obtenidos para confirmar que el algoritmo cumple su función de manera óptima.

Solución presentada

El algoritmo que se propone se basa en la Distancia de Hausdorff. Este concepto mide el nivel de discrepancia entre dos grupos de puntos de un mismo espacio métrico y puede cumplir con los requisitos deseables de invarianza para la detección de formas en imágenes con modificaciones triviales sin, además, introducir un gran coste computacional adicional. El algoritmo propuesto ofrece la posibilidad de paralelizar la parte más pesada de la computación; tarea que puede orientarse desde cualquiera de los dos principales paradigmas de paralelización: ya sea por Threads y tareas o por SIMD y sobre los datos.

1.4. Planificación

La estructura de este documento está de acuerdo con la ejecución cronológica de las fases del proyecto. La estimación de tiempos se ha realizado suponiendo una dedicación de 8h al día.

- Estado del arte:
  - Tecnologías de paralelización: 15 días.
  - Tecnicas de análisis de imágenes: 20 días.
- Desarrollo:
  - Estudio y creación de un algoritmo basado en la distancia Hausdorff: 10 días.
  - Diseño e implementación secuencial del programa: 15 días.
  - Implementación de la paralelización con OMP: 4 días.
- Pruebas y resultados:
  - Diseño del conjunto de pruebas: 1 día.
  - Ejecución de las pruebas: 2 días.
  - Análisis de los resultados y conclusiones: 3 días.
- Total: 75 días / 600 horas.

Este total de 600 horas ha estado repartido a lo largo de 10 meses, lo que da una media de 2 horas al día. El coste del proyecto solo se deriva de las horas de trabajo del ingeniero, ya que se ha desarrollado bajo Ubuntu Linux en C/C++ y con plataformas Open-Source. Sabiendo que la retribución de mercado es de 40€/h en ingenieria informática esto nos da un total de 24.000€.

Parte I

Estado del Arte

En esta sección se ofrece una visión general sobre las tecnologías existentes en el ámbito del presente trabajo: las herramientas principales de paralelización de código y las ideas y algoritmos típicos para la detección y reconocimiento de imágenes de dos dimensiones. Los apartados de la sección permiten situarse dentro del marco actual y nos indican el camino para realizar un proyecto que contribuya con ideas nuevas o mejoras de los trabajos anteriores.

Esta sección, en el contexto de la ingeniería de software, se corresponde con el análisis detallado del problema y de ella se pueden deducir una serie de requisitos básicos que debe cumplir la aplicación que se ha desarrollado.

Capítulo 2

Tecnologías de Paralelización

Existen múltiples tecnologías que facilitan la escritura y el desarrollo de programas capaces de realizar ejecución de procesos paralelos. Dentro de éstas se distinguen fundamentalmente dos aproximaciones: Threads y SIMD. A continuación se presentan una serie de herramientas y productos en cada una de estas categorías. Para una comparativa en mayor profundidad de algunas de estas tecnologías ver Sanchez et al. (2012).

2.1. Threads

En esta sección se describen las principales tecnologías de procesamiento paralelo basadas en hilos. Estas técnicas se basan en la paralelización de tareas, de forma que la carga de trabajo pueda repartirse entre distintos recursos. Según la filosofía básica de la paralelización con hilos cada tarea tiene su propia estructura de memoria y, en principio, el acceso a sus datos es privado, pero esto puede variar dependiendo de la tecnología que se utilice.

2.1.1. Hilos de C++ 11

El estándar C++ 11 introduce por primera vez soporte nativo de programación multi-hilo. Con esta novedad es ya posible crear programas de C++ multi-hilo sin necesidad de recurrir a extensiones externas dependientes de la plataforma. Esta funcionalidad se ofrece en la nueva biblioteca estándar de hilos de C++.

Hasta la fecha, la programación multi-hilo de C++ se basaba en el uso de bibliotecas creadas por terceros que agrupan la funcionalidad necesaria y proporcionan una buena abstracción de alto nivel, que permite a los programadores la creación de código paralelizado. Uno de los problemas fundamentales es que el estándar previo de C++ no reconoce la existencia de hilos de ejecución, por lo que el modelo de memoria no estaba formalmente definido para este tipo de aplicaciones. En la mayoría de los casos las bibliotecas como Boos y ACE han servido como base sólida para la programación paralela en C++, pero al no poseer un modelo de memoria que tenga en cuenta la existencia de hilos surgen problemas, especialmente en aquellas aplicaciones que tratan de lograr alto rendimiento haciendo uso de conocimientos específicos del hardware, o para los

desarrolladores creando programas multi-plataforma donde el comportamiento del compilador varía según la plataforma.

Esta situación ha motivado la creación, por parte del comité de estandarización de C++, de una biblioteca estándar para hilos en este lenguaje, donde se define un modelo de memoria adecuado para la práctica de esta programación y se resuelven otros problemas subyacentes. Esta nueva biblioteca ofrece cuatro funcionalidades básicas: clases para el manejo de hilos, mecanismos de protección para datos compartidos, operaciones de sincronización entre hilos, y operaciones atómicas de bajo nivel. La nueva biblioteca ha sido desarrollada basándose en la experiencia acumulada con las bibliotecas externas anteriormente mencionadas, especialmente se ha trabajado estrechamente la biblioteca Boost, tomándola como ejemplo e incluso manteniendo una estructura y nomenclatura similares. Gracias a esto los programadores experimentados en Boost se sentirán familiarizados usando el nuevo estándar de C++.

Es muy destacable la inclusión de las operaciones atómicas que permiten a los programadores optimizar su código abstrayendose de la arquitectura, de forma que no sea necesario incluir secciones de código optimizadas utilizando el lenguaje de ensamblador correspondiente. Con esto se consigue un código optimizado más portable, ya que será el compilador el encargado de realizar los ajustes necesarios sobre la maquina sin necesidad de modificar el código de partida.

Para conocer los detalles de estas nuevas funcionalidades incluidas en la biblioteca estándar de C++ y tener un punto de partida para aprenderlas a fondo, se recomienda consultar a Williams A. en Williams (2011).

2.1.2. OpenMP

OpenMP es un API ideado para la programación de tareas multiproceso con memoria compartida, es decir, una encapsulación de threads. OpenMP se encuentra disponible para los lenguajes C, C++ y Fortran en la mayoría de las arquitecturas y sistemas operativos. EL API consiste en una agrupación de directivas para el compilador, rutinas de biblioteca y variables de entorno que permiten modificar el comportamiento de la ejecución de un programa. En el caso de sistemas grandes, como clusters, OpenMP puede ejecutarse simultáneamente con MPI, de forma que el código pueda paralelizarse de manera sencilla en cada maquina del cluster con MPI, y dentro de cada procesador con las directivas de OpenMP.

El funcionamiento básico de OpenMP, como se explica en Barney (1998) se basa en la idea de *fork-join*, donde el hilo de ejecución maestro se divide en diversos hilos. Esta división debe efectuarse en las secciones de código que lo soporten, ya que de otra forma en lugar de mejorar el rendimiento, este empeoraría. Una vez realizada la ejecución en paralelo, los hilos se juntan de nuevo para continuar con la ejecución del hilo principal.

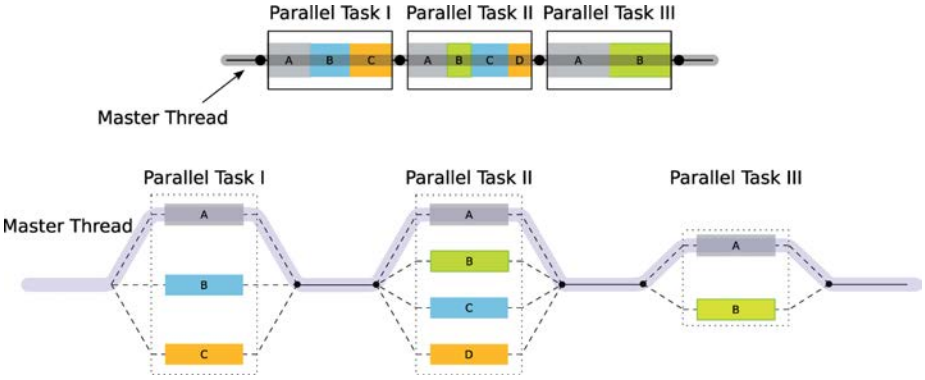


Figura 2.1: Paralelización en OpenMP

Cada hilo tiene su propio identificador, que puede usarse para mejorar la paralelización o comunicarse con un determinado hilo. Este modelo se basa en la memoria compartida, donde todos lo procesadores o threads tienen acceso a la misma sección de memoria. Esto hace que OpenMP proporcione cláusulas que se deben añadir a la directiva de OMP que permiten aislar ciertas variables o secciones de código dentro de una paralelización para evitar problemas de concurrencia o que puedan tener lugar condiciones de carrera, esto es, que el resultado no dependa del orden en que se ejecuten las tareas. Existen cláusulas de control sobre los datos y accesos a memoria, cláusulas de sincronización y cláusulas de planificación. Además existen cláusulas extras que añaden funcionalidad a OpenMP, como cláusulas de inicialización de variables, cláusulas para la copia de variables, la cláusula *reduce* para unificar resultados tras un *join* entre otras.

OpenMP ofrece una interfaz sencilla e intuitiva que permite una rápida paralelización del código, y potencia para desarrollar implementaciones más complejas que aprovechen mejor las ventajas de la ejecución simultánea. Por contra, al ser un API externo, puede generar problemas de sincronización difíciles de depurar, así como la imposibilidad de manejar manualmente la distribución de hilos en los recursos disponibles.

2.1.3. TBB

TBB, acrónimo de Threading Building Blocks, es una tecnología de la compañía Intel Intel (2012), que se presenta en forma de biblioteca de C++ para el soporte y desarrollo de programación paralela. Diseñada para promocionar la programación escalable de datos paralelos, tiene además soporte para paralelismo anidado, lo que permite construir bloques paralelos tomando como base otros bloques paralelizados. En TBB se especifican tareas, no hilos, y se deja que la biblioteca mapee dichas tareas en hilos de manera eficiente.

La biblioteca ofrece una funcionalidad similar a la de OpenMP, aunque de diferente ejecución. TBB contiene algoritmos básicos y avanzados de paralelización, contenedores de datos concurrentes, distribución de memoria escalable, controles de concurrencia, operaciones atómicas, herramientas de medición de tiempos y un planificador de tareas.

La aproximación de paralelización de TBB consiste en dividir las tareas de

forma equitativa entre los recursos disponibles. Si alguno de los recursos termina sus tareas mientras otros aún poseen una carga significativa de trabajo, TBB asignará parte de este trabajo, si es posible, al recurso que ha quedado libre. Este hecho permite, a una misma aplicación, escalar de forma diferente dependiendo de la máquina en que se ejecute sin necesidad de cambiar el código.

TBB se encuentra disponible tanto de forma comercial como en forma *open source*.

2.2. SIMD

En esta sección se detallan las principales tecnologías paralelas de procesamiento basadas en el concepto Single Instruction, Multiple Data o SIMD. La idea básica de esta técnica consiste en realizar la misma operación sobre múltiples datos de forma simultánea. Esto se puede lograr de diversas maneras. A continuación se exponen las principales, desarrolladas por diversas compañías.

2.2.1. CUDA

CUDA es una tecnología creada por la compañía nVidia que permite utilizar sus tarjetas gráficas para la realización de cálculos matemáticos. Esto es posible porque las imágenes que renderizan o tratan los procesadores gráficos se representan en forma de matrices, por lo tanto el paso que permite a un procesador gráfico usar esa potencia de cálculo para otras operaciones matemáticas más generales no resulta complicado. Este lenguaje es una extensión de C, por lo que resulta familiar e intuitivo, siendo además necesario estudiar bien la estructura de memoria de las tarjetas gráficas para así poder sacar el máximo provecho a esta tecnología.

A causa del diseño de las tarjetas gráficas, la manera de enfocar lo problemas a resolver usando CUDA debe ser necesariamente paralela. Ésta es la idea central en la que se basan los creadores de CUDA para proporcionar una herramienta de proceso de datos que sea más rápida y eficiente que un procesador multipropósito general.

Los procesadores gráficos que permiten el uso de CUDA no están diseñados usando la misma idea que un procesador de propósito general multinúcleo. De forma gráfica:

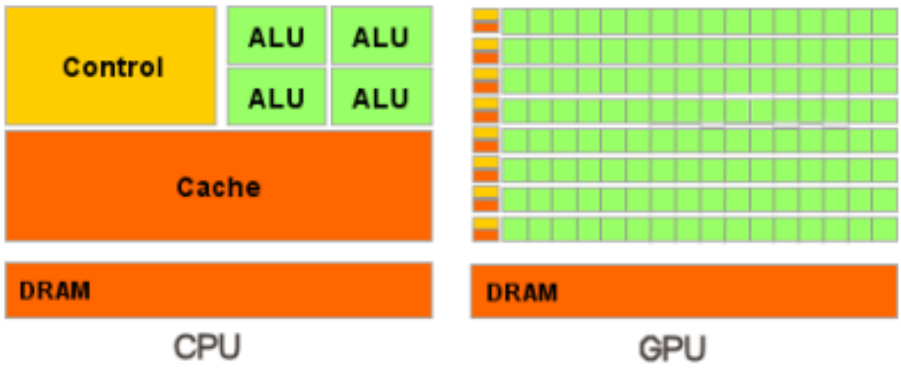


Figura 2.2: CPU vs GPU

Como puede verse en la figura 2.2, la arquitectura de las GPU se centra en tener más núcleos especializados únicamente en procesar instrucciones que realizan operaciones aritméticas. Éstos núcleos de procesamiento cuentan con relativamente poca memoria individual, frente a una gran cantidad de memoria compartida. Por contraste, las CPU se centran en tener menos núcleos, de forma que éstos puedan procesar a mucha más velocidad una mayor variedad de instrucciones complejas. De este enfoque se deduce que la capacidad de cómputo de un procesador gráfico viene de la segmentación de los algoritmos en cientos de unidades que pueden ejecutarse individualmente, para luego agrupar el resultado general.

El principal inconveniente del uso de tarjetas gráficas a la hora de hacer computación de alto rendimiento, radica en las transferencias de datos, ya que el bus entre los puertos PCIe donde se conectan las tarjetas y la memoria principal es mucho más lento que la conexión entre la CPU y la memoria. El resultado es que una operación pequeña tal vez se resuelva de forma más rápida en una GPU, pero debido al tiempo que se pierde en las transferencias de entrada y salida de datos es más eficiente hacer la misma operación en la CPU. Sin embargo para operaciones que pueden llevar un largo tiempo, sí que resulta rentable el uso de tarjetas gráficas, ya que en ese caso el tiempo de comunicación entre la memoria y la/s tarjetas es despreciable frente al de proceso. Los elementos de ejecución de CUDA son los siguientes nVidia (2012a):

- **Grid:** Un programa, que en CUDA se denomina *kernel*, genera un *grid* para poder ejecutarse. Los *grids* contienen bloques que pueden estar estructurados en una o dos dimensiones.
- **Bloque:** Los bloques son los componentes de los *grids* y agrupaciones de *threads*.
- **Thread:** Unidades básicas de ejecución. Cada *thread* ejecuta el mismo código, que será el del *kernel*. Éstos threads pueden tener una estructura de hasta tres dimensiones. Haciendo uso de las dimensiones en que estén organizados se puede diferenciar la funcionalidad de cada *thread* y los resultados generados.

En la siguiente figura se puede ver la estructura de un *kernel* básico, que

cuenta con un *grid* de 3x2 *bloques*, y cada uno de éstos contiene 4x3 *threads*, lo cual representa una capacidad total de 72 threads de ejecución.

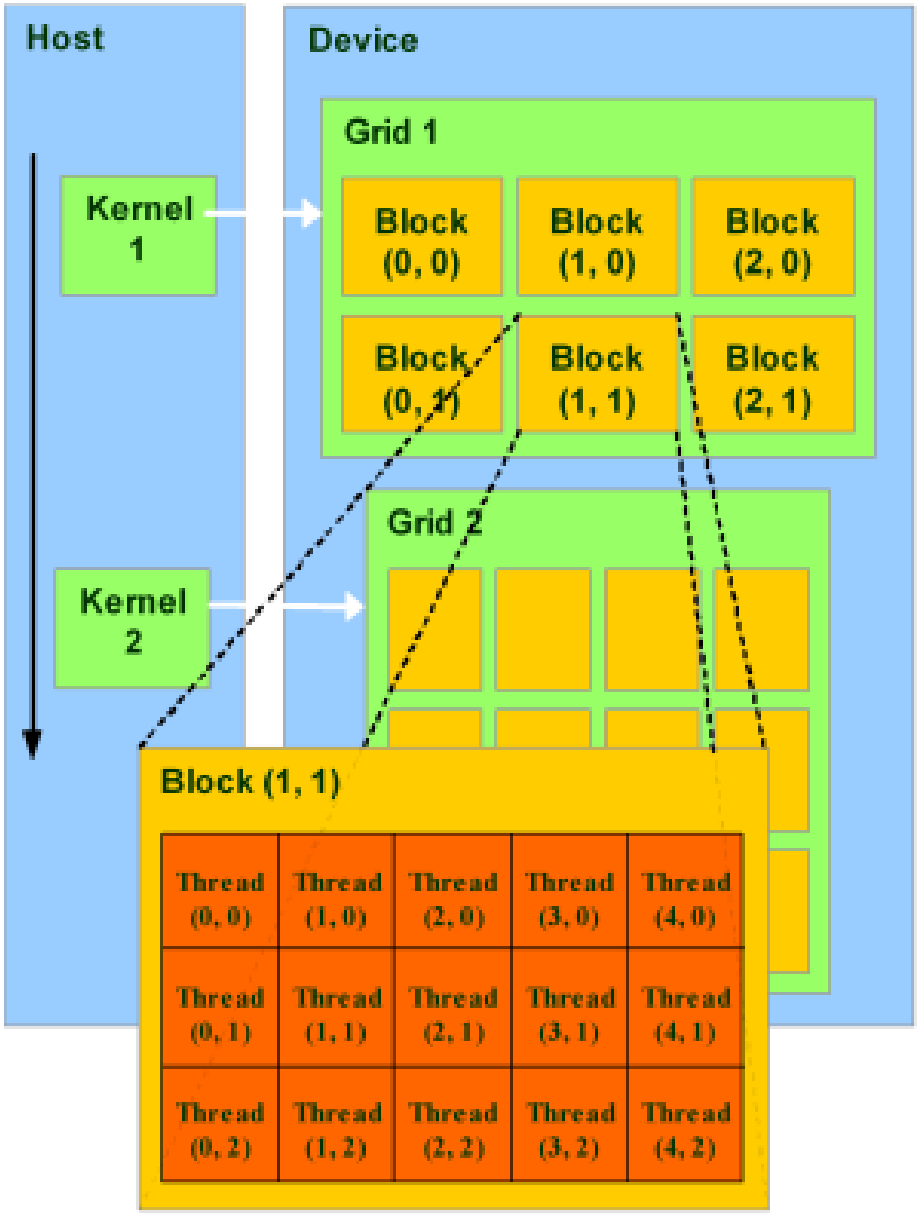


Figura 2.3: Organización de Hilos en CUDA

La última generación de procesadores gráficos lanzada al mercado por nVidia cuenta con 1536 *kernels*, en contraste con los 512 de la generación previa.

Las principales mejoras en la arquitectura de los procesadores de nVidia capaces de ejecutar código CUDA son: la disminución del tamaño de los transistores y el consumo energético. NVidia se esfuerza para introducir todas las

mejoras posibles de forma que la arquitectura básica sigue siendo la misma, para así ofrecer compatibilidad entre las versiones diferentes de CUDA.

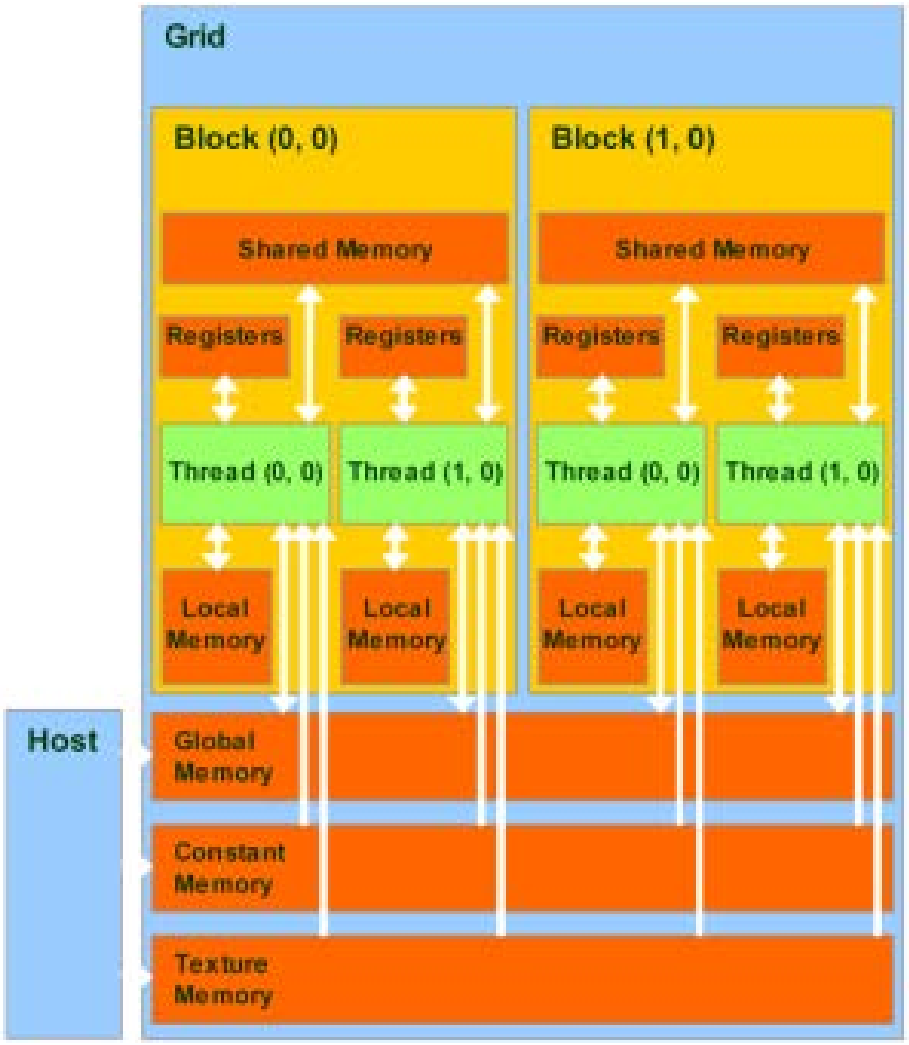


Figura 2.4: Estructura de memoria en CUDA

La figura 2.4 representa la estructura de memoria de las GPU mientras procesan programas en CUDA.

- **Global Memory:** Memoria cuya tarea es comunicar la CPU con la tarjeta gráfica. Como esta memoria no dispone de caché propio es necesario aprovechar al máximo el ancho de banda del BUS para hacer el mínimo de transferencias posibles entre la GPU y la CPU.
- **Constant Memory:** Otra memoria que se puede usar para la comunicación entre la CPU y la GPU. Esta memoria sí tiene caché, pero al ser de sólo lectura (por parte de la GPU) su uso debe ser cuidadoso, ya que una

vez la CPU escribe un dato en la memoria, permanece constante para la tarjeta.

- **Texture Memory:** Como su nombre indica, esta memoria está destinada al tratamiento de gráficos, y ha sido ideada para tener mejor ancho de banda a costa de ofrecer una menor latencia.
- **Local Memory:** Las memorias anteriores están compartidas por todos los GPCs (Graphics Computing Clusters). Sin embargo esta memoria es exclusiva para cada *thread* y se utiliza para sus variables locales.
- **Shared Memory:** Memoria compartida por todos los hilos de un bloque. Es la más rápida de la tarjeta y si se programa adecuadamente puede igualar la eficiencia del uso de registros en una CPU.

Todas las descripciones anteriores son a nivel lógico, ya que a nivel físico las GPU siguen estando diseñadas para el tratamiento de gráficos y generación de la salida de vídeo. En la figura 2.5 se muestra un ejemplo del diseño físico de la arquitectura llamada Kepler:

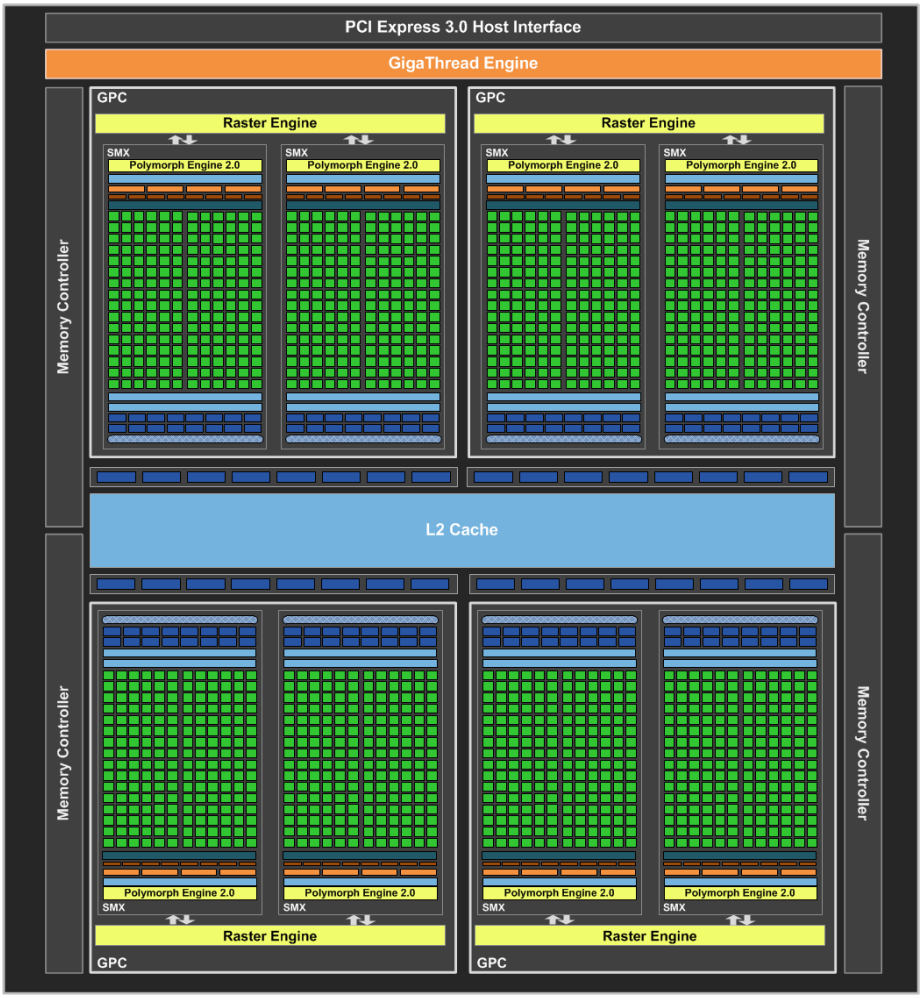


Figura 2.5: Procesador Kepler de nVidia

La estructura principal de proceso de Kepler son los SMX, que no son más que las agrupaciones de unidades de cálculo conectadas con elementos de entrada salida y sus diferentes niveles de memoria.

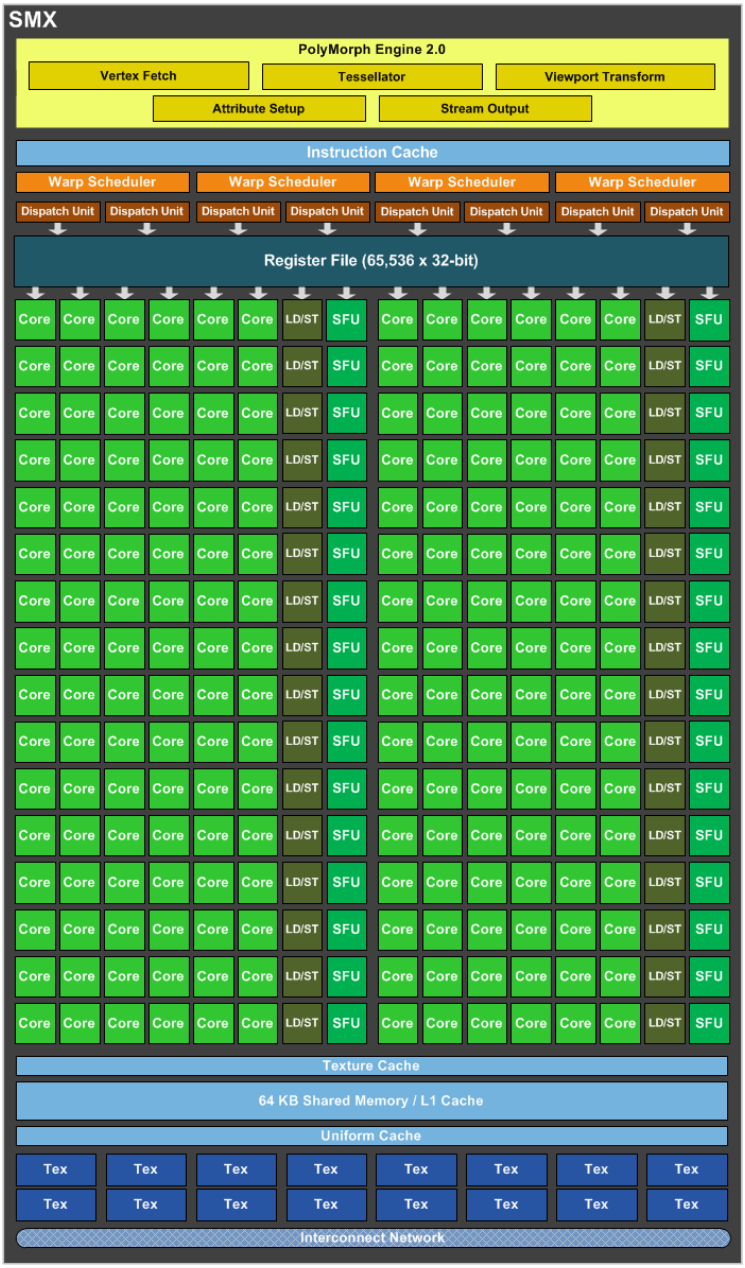


Figura 2.6: Detalle de los SMX en Kepler

Puede verse en la figura 2.6 que la estructura en memoria descrita anteriormente encajará de forma aproximada con la arquitectura física, pero ésta se salta los componentes específicos para el tratamiento de gráficos.

Además de tener en cuenta la extraña estructura de memoria que plantea CUDA, el programador deberá familiarizarse también con la configuración de los *threads*. Previamente se ha explicado que cada bloque es una agrupación de *threads*, pero falta un componente intermedio: el *warp*. Un *warp* es la unidad

mínima de *threads* que se agrupan en un bloque. Ésta agrupación será de 32 *threads*, tamaño que no puede ser cambiado por el usuario. Ésto es importante porque un *thread* es la unidad mínima de ejecución, pero un *warp* es la unidad mínima que puede manejar el programador, lo que conduce a que el uso de barreras de sincronización no se hace a nivel de *thread*, sino a nivel de *warp*. Además, esta agrupación obliga a que todos los *threads* de un *warp* deben realizar la misma tarea.

Otra limitación importante es que el tamaño máximo de un bloque es de 1024 hilos, y que por cada dimensión tienen un máximo de: 1024 en X, 1024 en Y y 64 en Z; esto quiere decir que en cada bloque se pueden tener configuraciones del tipo 128 x 4 x 2, 4 * 64 * 4, etc. El tamaño máximo de un *grid* es de 65536 por cada dimensión, dando un máximo de 64 bloques de 1024 *threads*. Éstos datos son ciertos para la versión 4.1 de CUDA nVidia (2012b), pero seguramente crecerán en el futuro.

2.2.2. OpenCL

Éste lenguaje tiene los mismos objetivos que CUDA: permitir derivar cálculos matemáticos a tarjetas gráficas, pero con la diferencia de que OpenCL trata las plataformas de forma homogénea. Esta característica, que permite hacer uso de distintos fabricantes de tarjetas y CPUs en la misma máquina, se deriva de su concepción como estándar gratuito y abierto. OpenCL ha sido desarrollado por el grupo Khronos, que ha contado con la colaboración de Intel, Apple, ATI y nVidia.

El enfoque de OpenCL, pese a no estar orientado a ninguna arquitectura concreta, permite un control a más bajo nivel, con lo que se puede lograr un mayor rendimiento a costa de una mayor dificultad en su programación. Además, el código de OpenCL es totalmente portable, de forma que un mismo programa compilado puede ejecutarse sin problemas sobre máquinas con múltiples CPUs, máquinas con cluster de tarjetas gráficas, etc. de forma que opera como si fuese una única plataforma.

La arquitectura lógica de OpenCL se basa en los dos siguientes elementos Group (2011):

- **Hosts:** Servidores que contienen dispositivos compatibles con OpenCL, que en la figura 2.7 se corresponde con el Compute Device. Una aplicación de OpenCl se ejecuta en el *host*, que es el encargado de realizar el reparto de tareas en los dispositivos disponibles.
- **Compute Units:** Cada dispositivo que forma parte del *host* puede tener a su vez una subdivisión en componentes capaces de realizar cálculos, como puede ser los núcleos de un procesador o las unidades de calculo de una GPU. Éstos son los elementos que se encargan de realizar todas las operaciones, procesando cada uno de ellos un único flujo de instrucciones.

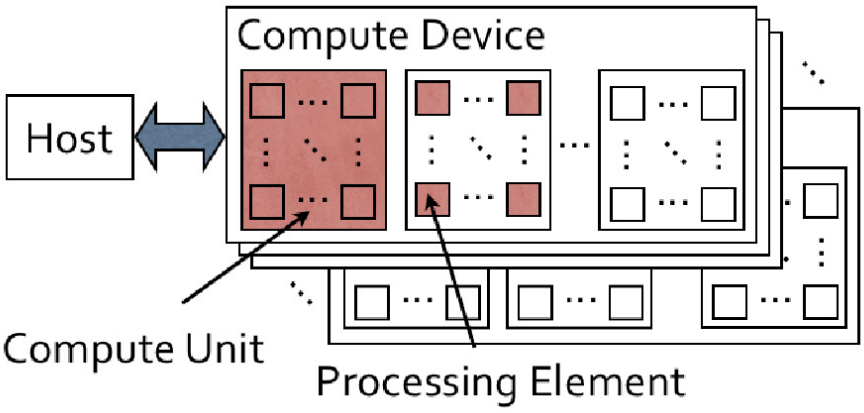


Figura 2.7: Oranización de OpenCL

La ejecución de las aplicaciones de OpenCL se realiza en dos procesos. Primeramente el código anfitrión, que se ejecuta en el *host* y que es el encargado de gestionar los otros procesos, que al igual que en CUDA se llaman *kernels*. Éstos *kernels* son procesados por los Computing Devices. La organización interna de los *kernels* es similar a la explicada anteriormente en CUDA: cada *kernel* se divide en un espacio de índices donde cada índice se corresponde con un *work-group*. A continuación, cada *work-group* se separa en *work-items*. Éstos *Work-items* son equivalentes a los *threads* de CUDA, en el sentido que cada uno ejecuta el mismo código pero con datos deferentes (la filosofía base de SIMD). Los *work-items* tienen sus propios índices locales dentro de su *work-group*, y sus índices globales. Todos los *work-items* de un *work-group* se ejecutan de manera concurrente en un elemento de procesamiento. Los *work-groups* se organizan en *grids* que serán otro espacio de índices. Cada *Kernel* genera un único *grid* de hasta tres dimensiones, como se muestra en la figura 2.8.

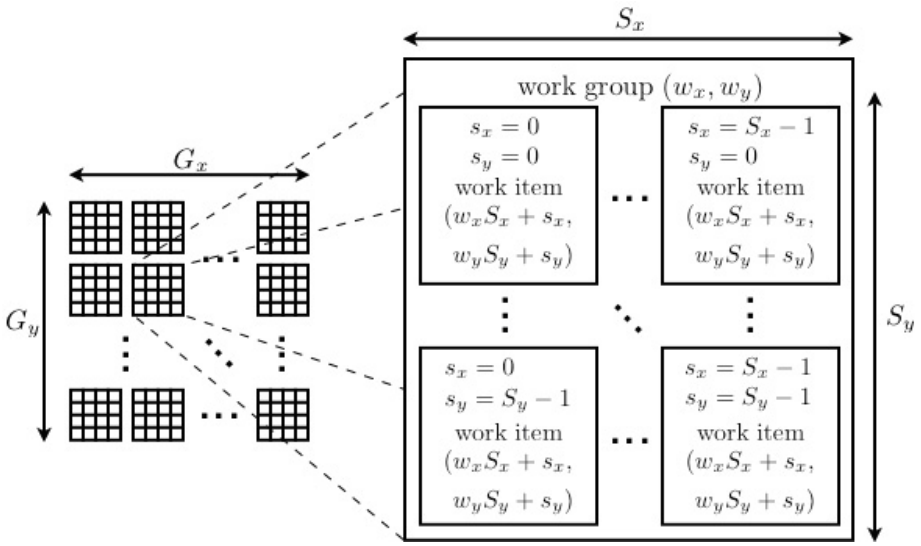


Figura 2.8: Estructura de los *Work-group*

Para permitir el procesamiento y que el *host* pueda repartir los *kernels*, OpenCL hace uso de la siguiente estructura de memoria AMD (2011):

- **Memoria Global:** Memoria principal a la que todos los *work-items* y el *host* tienen acceso pleno. Esta memoria podrá hacer uso de cachés asociadas si el dispositivo dispone de ellas.
- **Memoria Constante:** Almacena los elementos constantes durante la ejecución de un *kernel*. Solo el *host* tiene permisos de escritura en esta memoria.
- **Memoria Local:** Cada *work-group* dispone de su propia memoria especifica, a la que tienen acceso total y único. Suele usarse para guardar las variables locales del *work-group*, pero también puede emplearse para mapear secciones de la memoria global.
- **Memoria Privada:** Memoria exclusiva de un *work-item*. Las variables almacenadas en este espacio no podrán ser accedidas por ningún otro elemento.

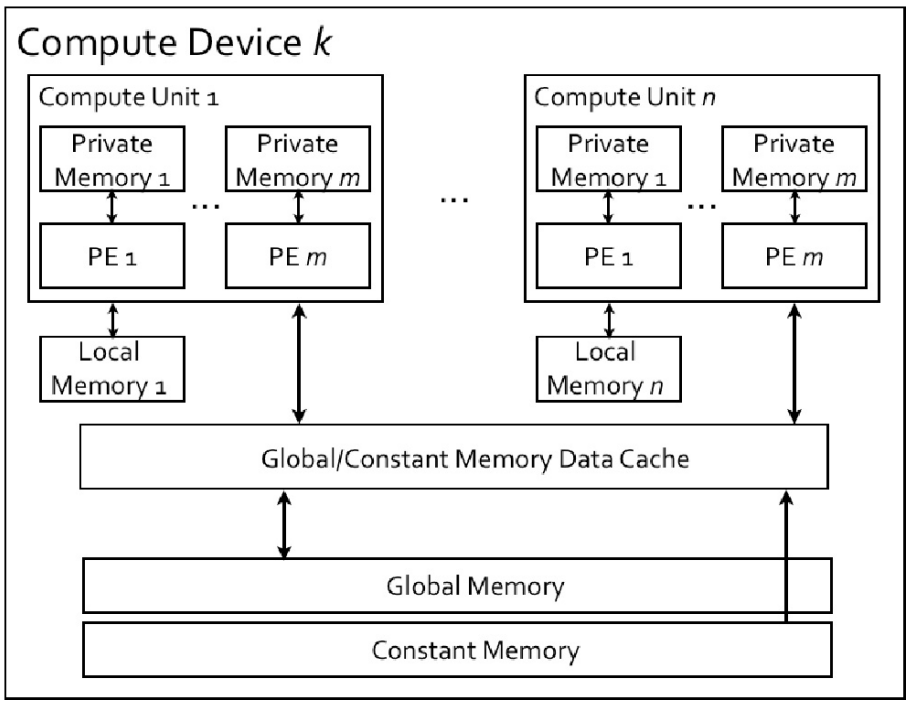


Figura 2.9: Mapa de memoria de OpenCL

Gracias a su modelo homogéneo, OpenCL ofrece tres posibilidades a la hora de realizar la programación concurrente: paralelización de datos, paralelización de tareas y una combinación de ambos. La paralelización de los datos es equivalente al modelo usado en CUDA, donde el programador puede gestionar personalmente la agrupación de *work-items* o indicar el total de éstos y permitir que OpenCL haga la división de forma autónoma. La segúnda opción es similar a la programación clásica de *threads* donde se separan las tareas que son independientes y se ejecutan de forma paralela. La última opción consiste en dividir las tareas y gestionar manualmente los *work-items* de cada una de ellas.

Al ser un lenguaje que pretende ofrecer una interfaz homogénea para realizar cómputos en sistemas heterogéneos, hay que tener muy presente las capacidades del sistema en cuestión, para poder obtener la máxima aceleración, aunque de esta forma siempre se ha de sacrificar la portabilidad del programa. OpenCL además tiene la misma desventaja que CUDA: la conexión entre los diversos dispositivos depende de la velocidad de los buses, y por lo tanto puede perderse una gran cantidad de tiempo en transferencias de datos si éstas no son las mínimas posibles.

2.2.3. ArBB

ArBB o Intel Array Building Blocks, es la solución presentada por Intel para resolver problemas de paralelización en sistemas heterogéneos, donde se pueda aprovechar la capacidad de proceso tanto de CPUs como de GPUs y procesadores *many core* que Intel planea sacar al mercado en el futuro próximo. ArBB ofrece un modelo basado en la composición dinámica de patrones estructurados

de programación paralela, ésto es, estructuras de código predefinidas que permiten ejecutar código de forma paralela de forma independiente al compilador.

Como presentan en Anwar Ghuloum (2010), la filosofía de diseño detrás de ArBB es la de permitir al desarrollador pensar en paralelo sin tener que preocuparse por los detalles de bajo nivel de su aplicación, como puedan ser los hilos de hardware, mapeo de éstos en los recursos disponibles, etc. ArBB proporciona un motor de ejecución dinámico que agrupa tres servicios:

- ***Threading Runtime***: Mecanismo que adapta dinámicamente la carga de trabajo a la arquitectura de la máquina en que se ejecuta. Proporciona modelos de paralelización de grano fino para datos y tareas. Además, se encarga de gestionar complejos patrones de sincronización.
- ***Memory Manager***: Es el encargado de distribuir, formatear y, con ayuda del *Threading Runtime*, particionar los datos para las operaciones paralelas. Incluye un recolector de basura y un conjunto de interfaces de liberación-cierre de memoria.
- ***Just-in-time Compiler/Dynamic Engine***: Esta tecnología genera una representación intermedia de las computaciones que van a realizarse, realiza optimizaciones, y genera el código que será finalmente ejecutado. La compilación sólo sucede cuándo es necesario; de forma que si existe tal compilación, el código se consigue de una caché de código en estado de precompilación.

Según Intel, ArBB puede entenderse de tres maneras deferentes:

- **Un modelo de Programación**: Para expresar paralelismo de datos con semántica secuencial. ArBB permite que las operaciones puedan expresarse sobre colecciones agregadas en lugar de sobre elementos individuales.
- **Un lenguaje**: ArBB añade nuevos tipos de datos y operaciones imitando el control de flujo de C/C++, además de incluirse en éstos mediante cabeceras y una biblioteca de ejecución.
- **Una máquina abstracta**: El modelo de ArBB se abstrae de los detalles de la máquina sobre la que se ejecuta, por lo que permite al programador realizar código paralelo portable de forma sencilla.

2.2.4. Cilk

Este lenguaje diseñado por el Laboratorio de Ciencias de la Computación del MIT y desarrollado de forma comercial por Cilk Arts, y posteriormente por Intel, forma una herramienta de programación de propósito general orientado a la programación paralela multi-hilo. El principio de diseño que sigue Cilk es que el programador debe ser el responsable de encontrar el paralelismo, identificando aquellos elementos de su algoritmo que pueden ser ejecutados de forma segura en paralelo; pero es el planificador de tareas el que realmente gestiona y decide en tiempo de ejecución como dividir el trabajo en los procesadores disponibles. Esta característica permite ejecutar un programa escrito en Cilk en procesadores con cualquier número de núcleos sin tener que cambiar el programa.

Cilk está basado en ANSI C con la adición de ciertas sentencias exclusivas del lenguaje for Computer Science (1998). Es por ésto que si, en cualquier programa escrito en Cilk, se eliminan las sentencias exclusivas del lenguaje, el resultado es un programa C válido. Ésto es solo cierto para la versión original de Cilk creada por el MIT, ya que primeramente la empresa Cilk Art desarrolló una versión llamada Cilk++ que soporta C/C++. Posteriormente Intel adquirió dicha compañía y desde entonces es el encargado del mantenimiento y mejora del lenguaje.

La paralelización en Cilk se realiza mediante las palabras reservadas cilk, spawn, sync, inlet y abort en la versión original del MIT.

- ***cilk***: Identifica una función escrita en Cilk. Es necesaria para distinguir el código Cilk de código C estándar.
- spawn: Indica que el procedimiento al que se le asigna puede ser ejecutado con seguridad en paralelo. Únicamente proporciona al planificador la información de que puede ser ejecutado en paralelo, no de que deba serlo.
- ***sync***: Sincroniza los procesos en paralelo. Indica que el proceso no puede continuar hasta que no hayan terminado de ejecutarse el resto de procesos.
- ***inlet***: Indica al procedimiento paralelizado que debe escribir los resultados de forma atómica en una variable interna de Cilk no accesible por otros hilos.
- ***abort***: Solo se puede usar en un proceso que haya sido declarado como *inlet*. Indica que el resto de procesos que se hayan formado por el padre pueden abortarse de forma segura.

El planificador de Cilk utiliza una política llamada 'robo de trabajo', en la que asigna tareas pendientes a los recursos que terminan con la suya siempre que queden tareas por realizar en la cola de trabajos. El planificador puede eliminar tareas de la cola de trabajos de un procesador y asignarlas a otro que haya terminando con las que tenía pendientes.

La versión actual que proporciona Intel mantiene la filosofía básica de Cilk, pero simplifica el lenguaje y añade extensiones para matrices, así como compatibilidad con C++ y una variedad de compiladores Intel (2009). El objetivo de Intel es convertir Cilk en un estándar de la industria .

2.3. Requisitos tecnológicos

Con la base adquirida sobre las tecnologías existentes, se proponen los siguientes requisitos para las herramientas de paralelización:

- La paralelización debe ser independiente de la máquina en que se ejecute una vez compilado el programa.
- Las herramientas deben ser gratuitas o disponer de versiones *Open-Source*.
- Es deseable que todas las herramientas puedan usarse en el mismo lenguaje.
- Es deseable que la herramienta permita realizar la parelización sin necesidad de reescribir el código secuencial.
- Es deseable que las tecnologías sean independientes del sistema operativo.

Capítulo 3

Análisis de Imágenes

Se presentan en este capitulo la descripción de una serie de técnicas y métodos de análisis de imágenes en dos dimensiones, que son ampliamente usados en programas de análisis, visión artificial y robótica. El análisis de imágenes es el proceso de obtención de información derivada de representaciones gráficas de cualquier tipo, ya sean estáticas planas de dos dimensiones, en cualquier formato (vectorial o un mapa de bits), en movimiento (secuencia de imágenes que forman un vídeo) o en tres dimensiones, ya sea por composición de los formatos anteriores o cualquier otro tipo de representación posible. Existen muchos métodos de análisis de imágenes, desde los clásicos a simple vista extrayendo conclusiones directas sobre la imagen, hasta los modernos computerizados, que ofrecen un espectro más amplio de posibilidades acerca de qué tipo de datos quieren obtenerse. En esta sección se estudian deferentes técnicas de extracción de información por ordenador sobre representaciones de imágenes como mapas de bits de dos dimensiones en color o en blanco y negro.

Gran parte de la información presentada en esta sección se basa en el trabajo de A. Ashbrook (1998)

3.1. Representación de invariantes

La representación de invariantes no es un método, sino una característica deseable de cualquier sistema de reconocimiento de imágenes, y pieza clave para las técnicas de aprendizaje automático. La idea base consiste en obtener un grupo compacto de descriptores. Para que un grupo de descriptores pueda ser considerado compacto, es necesario que mediante esa colección de parámetros sea posible reconstruir o reconocer el objeto del que se han obtenido de forma inequívoca. Puede establecerse que un conjunto de descriptores no compacto también resulte útil si ofrece información general suficiente. Las propiedades invariantes que se consideran elementales en el reconocimiento de imágenes, y en la visión en general, son la traslación, la rotación y la escala. Estas características permiten reconocer objetos en imágenes de forma independiente a la vista que se tenga de ellos.

La obtención y reconocimiento de los invariantes es un problema abierto y no trivial. El primer problema que se plantea es el poder descriptivo de estas características, ya que la representación debe contener información suficiente

como para poder discriminar entre conjuntos de objetos afines pero no iguales. El segundo problema consiste en la discriminación del ruido que puede producirse a la hora de obtener las imágenes o las manipulaciones que se realicen sobre ellas al tratar de obtener los invariantes.

Existen muchos algoritmos que permiten el reconocimiento de objetos en imágenes planas, muchos de los cuales se presentan a continuación en esta sección. No se ha explorado el reconocimiento de imágenes 3D u otras opciones, ya que éstos métodos están fuera del ámbito de este trabajo.

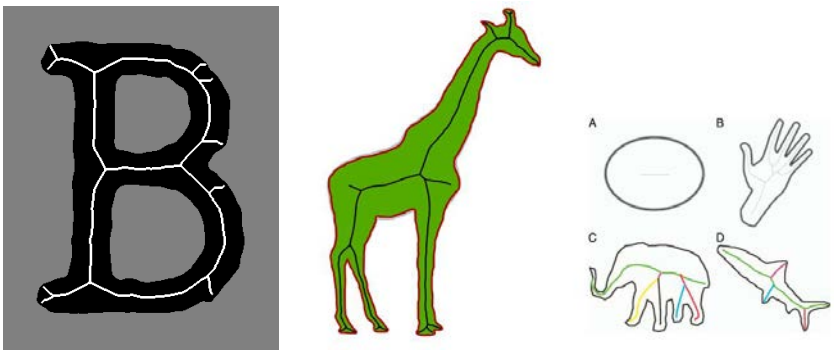
3.2. Block Matching

Las estrategias de esta técnica para el reconocimiento de un objeto en una imagen se realizan mediante la correlación de un patrón del objeto con la escena donde se espera encontrar el objeto, y se busca un umbral significativo que índique si hay o no una coincidencia. Dependiendo de la técnica concreta que se use, puede que las diferencias en la escala de colores, intensidad y saturación (el patrón puede estar en blanco y negro, diferente iluminación, etc) provoquen falsas conclusiones. Este problema puede solventarse si el reconocimiento se realiza solo con el contorno del objeto, en cuyo caso se pierde el resto de información de la imagen, que con técnicas adecuadamente robustas podrían ayudar a una mejor comparación.

De por sí, esta técnica no presenta de forma inherente la determinación de los invariantes antes mencionados, ya que depende de la técnica concreta de Block Matching que se utilice.

3.3. Esqueletos topológicos

La idea base tras este método se basa en la concepción de que toda la información relevante de una figura se encuentra en su topología básica. El esqueleto de una forma binaria (es decir una forma contra un fondo sin variaciones de color) se obtiene reduciendo repetidamente la figura hasta obtener una serie de lineas conexas de un píxel de grosor, como se muestra en la siguiente imagen:



(a) (b)

Figura 3.1: Ejemplos de esqueleto

La idea del reconocimiento mediante esqueletos topológicos consiste en comparar los esqueletos de los objetos que se quieren reconocer con un patrón dado, de forma que se pueda discernir hasta que grado sus características topológicas son coincidentes.

Existen diversas técnicas de obtención de esqueletos topológicos, pero todas ellas deben respetar las conexiones de la figura. Si en algún caso queda algún esqueleto no conexo con el resto -no puede quedar ninguna estructura aislada de la figura original, si el original es solamente una figura-, las propiedades topológicas se habrán perdido y la técnica de obtención de esqueletos no será válida.

Por definición, los esqueletos topológicos poseen todas las características invariantes deseables en cualquier sistema de reconocimiento de imágenes, pero esta descripción topológica puede resultar muy ambigua en situaciones reales donde existen más características que determinan la igualdad o no de dos objetos. Estas ambigüedades surjen del hecho que, si dos objetos comparten la misma estructura topológica, aunque sean objetos deferentes, matemáticamente serán considerados el mismo elemento. Éstos algoritmos presentan además problemas a la hora de reconocer objetos no completos o parcialmente cubiertos, ya que su esqueleto será demasiado diferente al del patrón que se está utilizando.

3.4. Momentos invariantes

Esta técnica se nutre de la definición del momento de una figura binaria de $M \times N$ dimensiones:

$$u_{pq} = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} i^p j^q f(i,j) \tag{3.4.0.1}$$

Donde $f(x,y)$ es la intensidad del píxel (1 o 0) en las coordenadas (x,y) y $p+q$ es el orden del momento. La expresión representa una función de la distancia entre los píxeles de la figura, por lo que el origen de las medidas se toma en relación al centroide de la figura para evitar variabilidad por traslaciones.

Las medidas individuales de éstos momentos no poseen la capacidad descriptiva necesaria para representar inequívocamente cualquier forma, ni poseen las características invariantes deseables, pero pueden definirse agrupaciones de funciones que tomen como base éstos momentos que sí posean esas características.

La clasificación de imágenes se realiza por comparación de vectores de momentos pertenecientes a imágenes con vectores conocidos de los patrones de la forma que se desea reconocer. Este método se puede forzar para ser invariante a la escala mediante un proceso de normalización de los vectores.

El método tiene mucha potencia, y es altamente viable para sistemas con aprendizaje automático, pero es muy sensible al ruido en las imágenes y a la ocultación parcial de las figuras a reconocer en la imagen, por lo que no se considera especialmente robusto.

3.5. Distancia Hausdorff

Como Huttenlocher y sus colegas presentan en Huttenlocher et al. (1993), la idea tras la distancia Hausdorff es que puede determinarse una distancia d_H

que indica el grado de disparidad entre conjuntos de puntos pertenecientes al mismo espacio. Si la distancia vale 0 ambos grupos serán el mismo, y cuanto mayor sea el valor, más diferentes serán entre sí. Esta propiedad puede usarse como base para el análisis de imágenes en formato mapa de bits, ya que, en este caso, las coordenadas de los píxeles forman los elementos del espacio y es posible comparar grupos que pertenezcan a éste.

Para que resulte útil como método de reconocimiento es necesario utilizar subconjuntos de los píxeles que forman las imágenes a estudiar, ya que si se comparan la totalidad de los píxeles y ambas imágenes poseen la misma resolución, la distancia valdrá 0, y por tanto no se obtendrá ninguna información. Esto se debe a que, en la concepción original de Hausdorff, la única propiedad que se utiliza es la geométrica, en este caso, las coordenadas de los píxeles. Sería posible redefinir la distancia para realizar la comparación entre la saturación del color o alguna otra característica de las imágenes. Debido a esta limitación, es común aplicar la distancia de Hausdorff para obtener la medida de la similitud entre dos contornos, como pueda ser el reconocimiento facial o la búsqueda de elementos en una imagen, ya que para estos casos se obtienen buenos resultados usando unicamente la información geométrica.

La principal ventaja de la distancia Hausdorff es que no es necesario que ambos grupos tengan la misma cardinalidad, o sea, el mismo número de elementos. Esta propiedad, aplicada al análisis de imágenes, permite que mediante el uso de esta técnica puedan determinarse equivalencias entre formas con diferente número de píxeles, lo que hace que resulte especialmente resistente al ruido y a la ocultación parcial de aquello que se quiere reconocer en una imagen. Por definición no posee las propiedades de invarianza deseables, pero éstas se pueden lograr con pequeñas transformaciones sobre los datos. La propiedad de traslación se consigue comparando figuras cuyo origen de referencia sea el propio centroide de la figura o elemento equivalente, como un vértice común. La invarianza de rotación y escalado puede lograrse multiplicando las posiciones de cada píxel por una constante, cuya dificultad de obtención depende exclusivamente de las imágenes objeto de estudio, aparte de otras mejoras como se presentan en Suau (2005).

3.6. Mapeo logarítmico-polar

Tomando un punto z del espacio de la imagen como $z = x + yi$, se puede mapear a un punto w del espacio log-polar de forma que $w = ln(z)$ y $w = ln(|z|) + i\theta_z$, y de esta forma podemos representar cualquier imagen en el espacio log-polar. La representación es intrínsecamente invariable frente a escala u orientación de la forma que se quiere reconocer en el espacio de la imagen, ya que estas transformaciones se correlacionan con una traslación en el espacio log-polar.

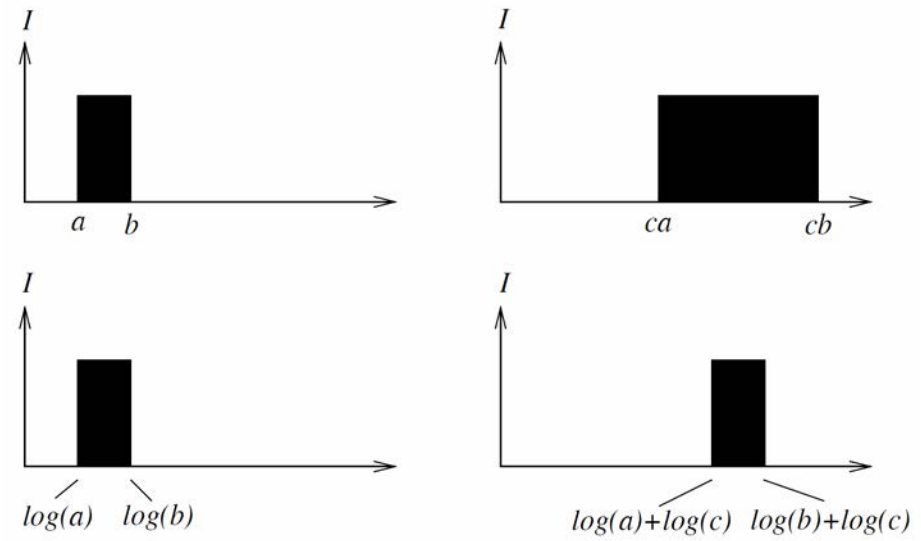


Figura 3.2: El escalado equivale a una traslación en el espacio logarítmico.

Para hacer el método más robusto, se ha propuesto el uso de la transformada de Fourier del espacio log-polar y utilizar su magnitud como invariante. Esto es posible porque la medida de la transformada de Fourier es invariante a traslaciones.

El método es intrínsecamente dependiente de la posición de la figura que quiere detectarse dentro de la imagen, por esto en la literatura se propone como solución localizar el eje de coordenadas en el centroide de la figura. Hacer esto elimina el problema de la traslación, pero es altamente sensible al ruido en las imágenes, y por lo tanto es sencillo que se obtengan resultados no deseados. Igualmente la singularidad de log(0) supone un problema, ya que objetos cerca de este punto tenderán a mostrarse apaisados en el espacio paramétrico, en lugar de desplazados, si son sometidos a traslaciones o rotaciones. Este problema puede solucionarse mediante imágenes con bordes muy determinados y confiando en que el objeto que se desea detectar no se localice cerca de dichos limites.

3.7. Descriptores geométricos de la forma

Dada una figura, existen una serie de sencillas medidas geométricas que pueden usarse para crear una representación de dicha figura. Estas medidas son las usuales: anchuras, alturas, perímetro, área, etc. Un descriptor de la figura será un vector o estructura de datos que contenga dichas medidas. La identificación de objetos se realiza por comparación con un vector descriptor conocido.

Esta técnica necesariamente necesita de un soporte que permita primeramente identificar la figura en la imagen, para poder luego obtener las medidas y así realizar la comparación. Por lo tanto, si el método seleccionado para reconocer formas no es robusto, sus errores se extenderán a los descriptores, aumentando la posibilidad de error en la detección. Por ello se recomienda el uso de múltiples sistemas de detección de figuras, de forma que se puedan obtener deferentes

medidas para la creación del descriptor, realizar medias y luego la comparación final. Se propone incluso el uso de esta técnica como método de verificación para discriminar errores con otros sistemas de reconocimiento.

3.8. Perfiles de frontera

Una forma eficiente de simplificar el proceso de comparación de contornos de objetos es describir dicho contorno como un perfil de una sola dimensión. Esto se puede conseguir de diferentes maneras:

3.8.1. Gráficas (r, θ)

Uno de los perfiles posibles es la representación del contorno en coordenadas polares relativas al centroide de la figura. Cada punto a lo largo del contorno se define por su distancia al centroide r, y su desplazamiento angular respecto a una referencia arbitraria, θ.

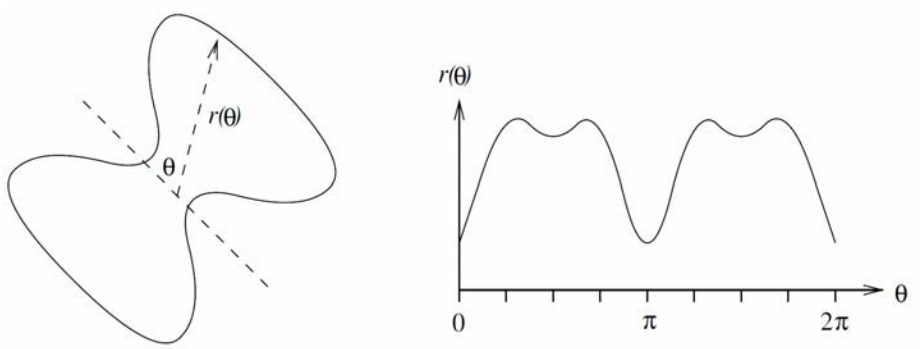


Figura 3.3: Figura plana y su representación (r, θ)

El reconocimiento se realiza mediante la comparación de los perfiles. Rotaciones sobre el objeto que se quiere detectar solo resultan en desplazamientos respecto al perfil patrón, por lo que la comparación se realiza desplazando un perfil sobre otro y encontrando una coincidencia suficiente.

El problema fundamental que plantea esta representación es que, para figuras complejas, existirán valores multievaluados de r para el mismo valor de θ. Esto quiere decir que el problema deja de ser unidimensional y vuelve a transformarse en uno bidimensional, por lo que se cuestiona la ganancia del uso de este método, ya que cualquier intento de solventar el problema (descartar valores de r en ciertas condiciones), reduce la capacidad descriptiva de la técnica, y por lo tanto su fiabilidad como sistema de reconocimiento.

3.8.2. Gráficas (s, ψ)

Este segundo método ofrece una opción para la representación de contornos que evita el problema de valores multievaluados y es relativamente más robusto al ruido que las representaciones (r, θ). Esta gráfica se genera comenzando en un

punto arbitrario del contorno, de forma que se representa la derivada de cada punto, ψ, frente a la distancia recorrida por el contorno, s.

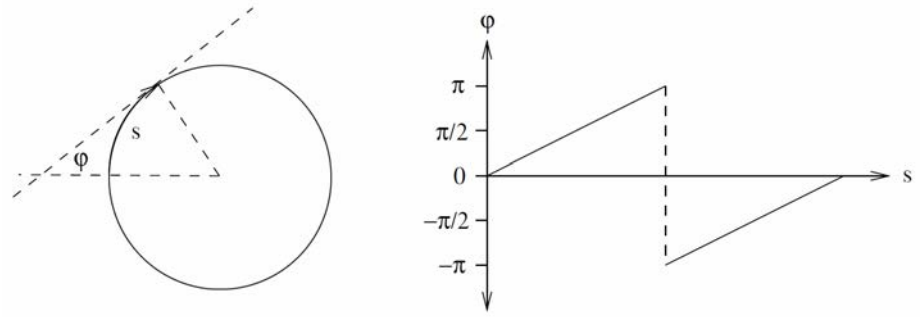


Figura 3.4: Figura plana y su representación (s, ψ).

En esta representación, una ocultación parcial del objeto a reconocer se traduce en la representación del contorno del objeto que produce la ocultación, dejando el resto del contorno intacto. Esto nos permite realizar un reconocimiento de objetos parcialmente cubiertos u ocultos comparando secciones de la representación.

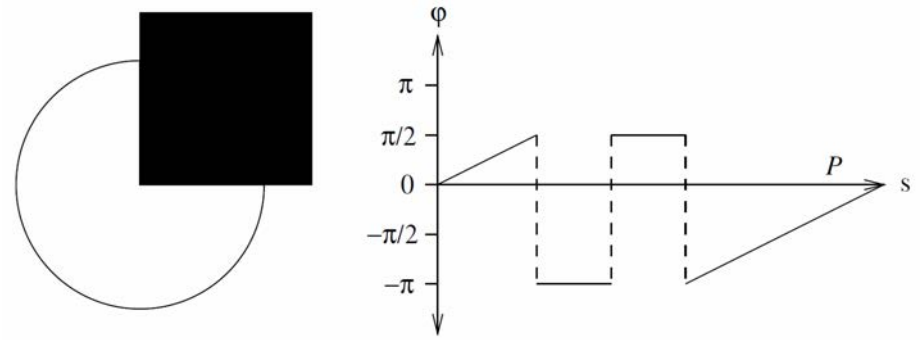


Figura 3.5: El cuadrado negro oculta parcialmente el círculo

3.9. Descriptores de Fourier

Tomando como base una de las representaciones anteriores del contorno de una figura, se puede normalizar a una distancia máxima de 2 y proceder a la obtención de sus coeficientes de Fourier mediante la siguiente expresión:

$$c_n = \frac{1}{N}\sum_{s=0}^{N} \Psi(s) e^{-i2\pi ns} \tag{3.9.0.1}$$

Tenemos ahora una representación univoca de infinitos coeficientes de Fourier que contienen la información del contorno. Los valores bajos de la serie describen el comportamiento macroscopico del contorno y los altos el comportamiento microscópico. En la practica el uso de infinitos coeficientes resulta

imposible, por lo que hay que truncar la serie para que resulte práctico. Esto es posible ya que por definición, haciendo uso del teorema de Nyquist, podemos muestrear una serie de Fourier y reducirla a un conjunto de valores finito sin perder información relevante.

El reconocimiento mediante este método es potente porque reduce la cantidad de información necesaria para proceder a la comparación. Sin embargo introduce una serie de cálculos que pueden ralentizar todo el proceso, además de heredar los problemas derivados de los perfiles de frontera que toma como base.

3.10. Modelos de Forma Activa

El modelo se basa en una distribución de puntos sobre el borde de la figura, sus vértices u otras características significativas del contorno. La posición de los puntos se define por las posiciones medias de éstos en un conjunto de imágenes de entrenamiento más una varianza, que describe cómo dichos puntos se desvían de su media en ligeras deformaciones producidas por las diferentes capturas del objeto en el conjunto.

El reconocimiento se realiza colocando los puntos sobre el contorno de la figura y ajustándolos a ésta. Si la varianza de estos puntos sobre el contorno a detectar entra dentro de los rangos del conjunto de entrenamiento, podemos decir que hay una coincidencia y el objeto ha sido satisfactoriamente detectado. Para que la detección sea posible hay que tener en cuenta los factores de escala, ya que el método es inmune a traslaciones o rotaciones, este problema se puede resolver mediante un coeficiente que ajuste las medidas del patrón a la escala del objeto que se trata de detectar.

El uso de este tipo de técnica está limitado por la necesitad de estimar inicialmente la posición y escala de los objetos a detectar en una escena de forma acertada, por lo tanto, si este paso previo puede realizarse satisfactoriamente, se puede decir que el objeto ya ha sido reconocido y tendremos el problema resuelto. Con ligeras modificaciones al método, como establecer un umbral al número de puntos que deben entrar en el rango de la varianza (no tienen porqué ser todos), este sistema de reconocimiento es bastante resistente al ruido y a las ocultaciones parciales del objeto a reconocer.

3.11. Transformada de Hough

Ideada inicialmente en física de altas energías para la detección de nuevas partículas, la transformada de Hough ha evolucionado y extendido su uso a diferentes aplicaciones de procesamiento de imágenes. Esta transformación se basa en la equivalencia de complejos patrones de píxeles del dominio de la imagen con grupos compactos en el espacio paramétrico elegido. La transformación opera de tal manera que agrupa conjuntos de puntos de la imagen a puntos singulares en el espacio parametrizado. La idea es tratar de reducir patrones complejos de píxeles en sencillos conjuntos parametrizados que resultan más sencillos de estudiar.

3.11.1. La Transformada de Hough en Línea Recta

Cualquier línea en una imagen puede describirse mediante su ecuación $y = mx + c$, y se representa como un solo punto en el espacio (m, c). De esta forma el problema de detección de rectas se simplifica a uno de detección de puntos. En la practica la transformación se aplica a cada punto de la imagen, que se verá transformado a una línea en el espacio paramétrico. Sin embargo, puntos de una misma recta en la imagen se mapean como una intersección de rectas en el espacio parametrizado. Cuando sucede una de estas acumulaciones, se puede decir que se ha encontrado un máximo local en el espacio parametrizado, y éstos máximos pueden usarse para determinar que se han detectado rectas en el espacio de la imagen.

Es necesario realizar modificaciones para poder tratar con rectas verticales o casi verticales, ya que en éstas el espacio paramétrico se vuelve infinito. Este problema se soluciona mediante el uso de dos espacios parametrizados, uno para rectas con pendiente menor que uno, el espacio original (m, c), y otro, (m', c') para líneas con pendiente igual o mayor que uno, donde $m' = \frac{1}{m}$ y $x = m'y + c'$.

La mayor dificultad de esta técnica es la enorme capacidad de computo requerida, ya que cada píxel de la imagen debe ser representado como una línea en el espacio paramétrico, y luego éste debe ser estudiado para encontrar los máximos locales. Existen mejoras para este método, pero quedan fuera del ámbito de este trabajo. Es necesario mencionar además, que este algoritmo solo es capaz de detectar la existencia de rectas en una imagen, pero para determinar la posición y longitud de estas requiere procesamiento adicional sobre la imagen.

3.11.2. Detección de círculos con la Transformada de Hough

En este caso el espacio paramétrico es congruente con el espacio de la imagen, esto es, cada punto de la imagen se mapea como un punto en la misma posición en el espacio paramétrico. Para detectar un circulo de radio R, otros círculos de este mismo radio se dibujan en el espacio paramétrico centrados en los contornos que se encuentren en la imagen. La intersección de éstos círculos indica el centro del circulo que se quiere detectar.

Al igual que la detección de líneas, este método es muy costoso, ya que hay que representar múltiples círculos en el espacio paramétrico para detectar uno solo de los círculos presentes en la imagen.

3.11.3. La Transformada de Hough generalizada

La Transformada de Hough generalizada no es más que una generalización de la técnica usada para la detección de círculos. La modificación consiste en que, en lugar de representar un punto en el espacio paramétrico a una distancia fija R a lo largo de la normal del borde actual, éstos puntos se representan ahora a una distancia variable $R(\theta)$ a lo largo de una línea que se encuentra desplazada de la normal un angulo variable $a(\theta)$, donde θ es el ángulo entre la normal y el eje x positivo. Una figura arbitraria será descrita por todos los $R(\theta)$ y $a(\theta)$ para todo θ en una tabla, usualmente referida como Tabla-R.

Si la forma definida en la Tabla-R se encuentra presente en la imagen, entonces cada sección del contorno de la forma añadirá a un punto L en el espacio paramétrco resultando así en una acumulación en dicho punto.

Al igual que los perfiles de frontera, esta aproximación se encuentra con el problema de los valores multievaluados para $R(\theta)$ y $a(\theta)$ para un mismo valor de θ en figuras complejas. En este caso resulta más sencillo solventar el problema ya que solo es necesario añadir éstos valores a la Tabla-R para tener perfectamente definida la silueta.

La Transformada de Hough generalizada no posee invarianza rotacional o de escala. Para ello hemos de crear un espacio de cuatro dimensiones donde dos de ellas representan la posición de la imagen, una la orientación y la última la escala.

3.12. Invariantes proyectivas 2D

Esta técnica se basa en el hecho de que los puntos de tangencia de objetos en un plano de dos dimensiones siempre se conservan bajo deferentes proyecciones, y que el mapeo de cualesquiera cuatro puntos de un plano a otro es suficiente para determinar la matriz de transformación T que defina completamente la transformación. Basta con mapear cuatro puntos de tangencia de un objeto plano, con otros cuatro puntos fijos pero arbitrarios de otro plano. El segundo plano poseerá las propiedades invariantes deseadas, y determinando la matriz de transformación T de los cuatro mapeos, todos los puntos pueden ser transformados del plano de la imagen al plano invariante.

Es común el uso de las concavidades de objetos planos para determinar cuatro puntos de tangencia. A los puntos así localizados se les llama puntos distintivos.

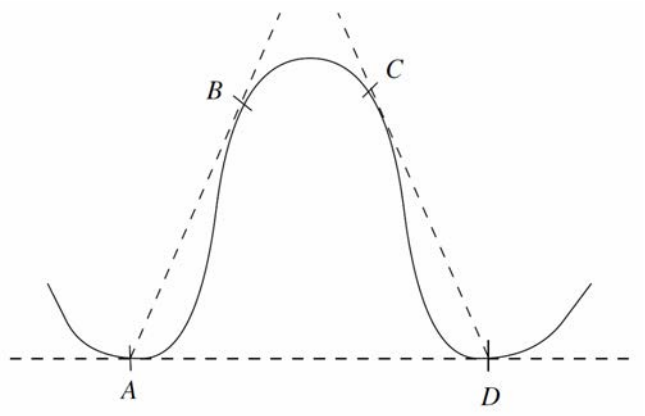


Figura 3.6: Puntos distintivos en una curva.

Dos punto distintivos (A y D) se localizan por la recta tangente que marca la entrada a la concavidad. Los otros dos puntos (B y C), se determinan por las tangentes con la curvatura interna de las rectas que pasan por los dos primeros puntos. Éstos cuatro focos se mapean como esquinas de un cuadrado de una unidad en el plano invariante, y el resto de puntos de la figura se mapearán de acuerdo a la transformación en este plano.

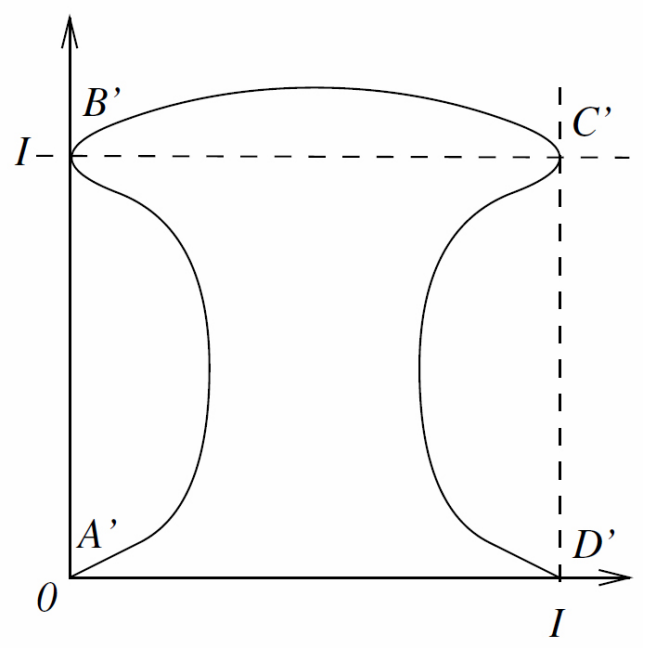


Figura 3.7: Mapeo de la figura anterior al plano canonico.

El reconocimiento de imágenes se hace mediante la obtención de vectores de invariantes basándose en medidas de las representaciones en el plano invariante y obteniendo un *hash* con dicha información. En el proceso de reconocimiento se descompondrán éstos vectores y se recompondrán los originales para buscar coincidencias sobre la imagen.

3.13. Requisitos del Algoritmo

Una vez enmarcado el problema, habiendo considerado las alternativas existentes y los objetivos del proyecto. se tiene que la solución que se propone debe cumplir con los siguientes requisitos del algoritmo:

- La técnica base deberá tener las propiedades de invarianza, o poder optar a ellas de forma trivial.
- Debe poseer procedimientos que sean fácilmente paralelizables.
- Es deseable que su coste computacional teórico no sea muy elevado.

Parte II

Desarrollo

En primer lugar se expone el concepto matemático de distancia Hausdorff, para posteriormente introducir el algoritmo propuesto para el reconocimiento de imágenes u objetos en imágenes. A continuación se presentan las herramientas que se han usado para implementar el algoritmo y cómo se ha realizado la programación. Finalmente se detallan las modificaciones necesarias de la programación en los procesos de paralelización que se han llevado a cabo.

Éste capitulo recoge los contenidos del diseño de la solución, partiendo de las necesidades detalladas en el capitulo previo.

Capítulo 4

Algoritmo propuesto

4.1. La distancia Hausdorff

El concepto central de éste trabajo es la distancia Hausdorff, idea introducida por el matemático Félix Hausdorff en su libro *Grundzüge der Mengenlehre*, (Fundamentos de Teoría de Conjuntos), en Abril de 1914 Hausdorff (1914). Ésta idea define una métrica que mide la distancia existente entre dos subconjuntos de un espacio métrico, es decir, la separación existente de dos grupos de puntos pertenecientes al mismo espacio. Como veremos, según esta medida, dos conjuntos están cerca el uno del otro si todos los puntos de uno de los grupos están cerca de alguno de los puntos del otro conjunto. Esta propiedad hace que la medida de la distancia no resulte simétrica, se tiene entonces que la distancia de un conjunto A a un conjunto B no tiene porqué ser la misma que la distancia del conjunto B al A.

La definición formal de la distancia Hausdorff viene dada por la siguiente expresión matemática:

Sean $A = \{a_1, ..., a_p\}$ y $B = \{b_1, ..., b_q\}$ dos conjuntos finitos no vacíos, subconjuntos de un espacio métrico (M, d), se define la distancia Hausdorff $d_H(A, B)$

$$d_H(A, B) = max\{h(A, B), h(B, A)\} \tag{4.1.0.1}$$

Donde

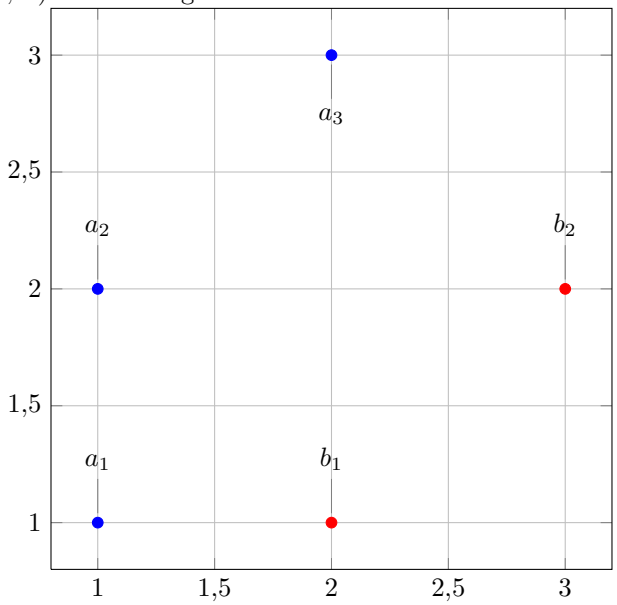
$$h(A, B) = max\, min||a - b|| \tag{4.1.0.2}$$
$$h(B, A) = max\, min||b - a|| \tag{4.1.0.3}$$

Con $a \in A$, $b \in B$ y $||\cdot||$ es una norma entre los puntos de A y B, como pueda ser la L_2 o la norma Euclídea.

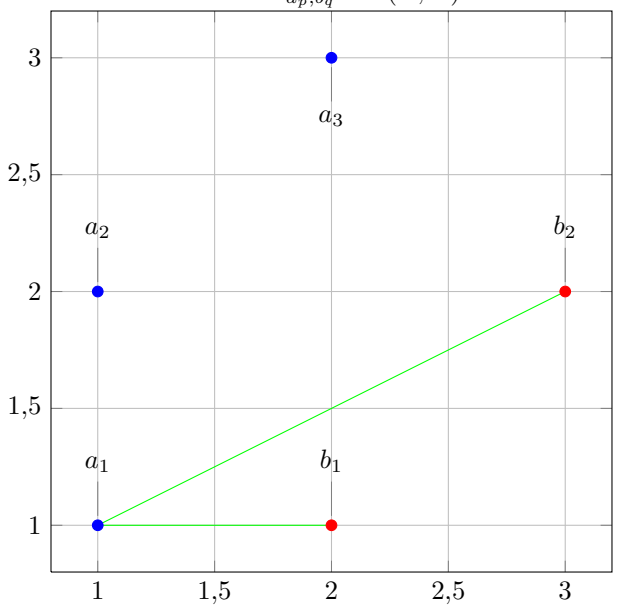
La función $h(A, B)$ se conoce como la distancia dirigida de Hausdorff, e identifica el punto $a \in A$ que está más lejos de cualquier punto de B y mide la distancia de a con su vecino más cercano en el conjunto B, usando la norma dada $||\cdot||$. Visto de otra forma, $h(A, B)$ genera una clasificación de todos los puntos de A basándose en sus distancias a sus vecinos más cercanos en B y toma como distancia entre A y B al primero de la lista. Intuitivamente, si $h(A, B) = d$,

entonces cualquier punto de A debe estar como máximo a d de algún punto de B, y existirá al menos un punto en A que se encuentre exactamente a d distancia del punto más cercano de B. La distancia $h(B, A)$ funciona de la misma manera pero eligiendo el punto de B que está más lejos de cualquier punto de A tomando la distancia con su vecino más cercano de este último.

La distancia Hausdorff $d_H(A, B)$ es entonces el mayor valor entre $h(A, B)$ y $h(B, A)$. De forma gráfica:

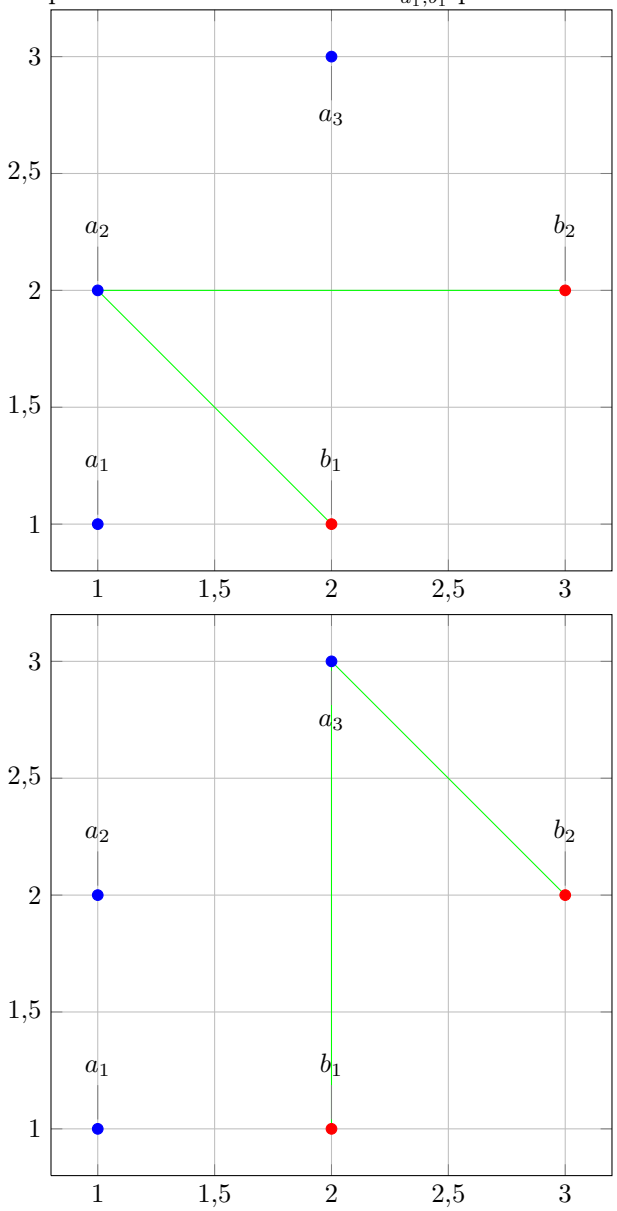


Sea siendo A el grupo de puntos de color azul $A = \{(1, 1), (1, 2), (2, 3)\}$ y B el grupo de puntos rojos $B = \{(2, 1), (3, 2)\}$. En los siguientes pásos se muestra la secuencia del cálculo de $d_{a_p,b_q} = h(A, B)$.

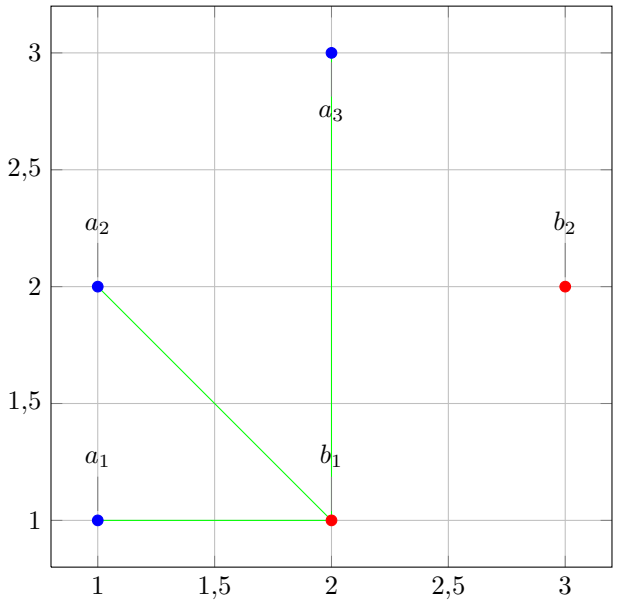


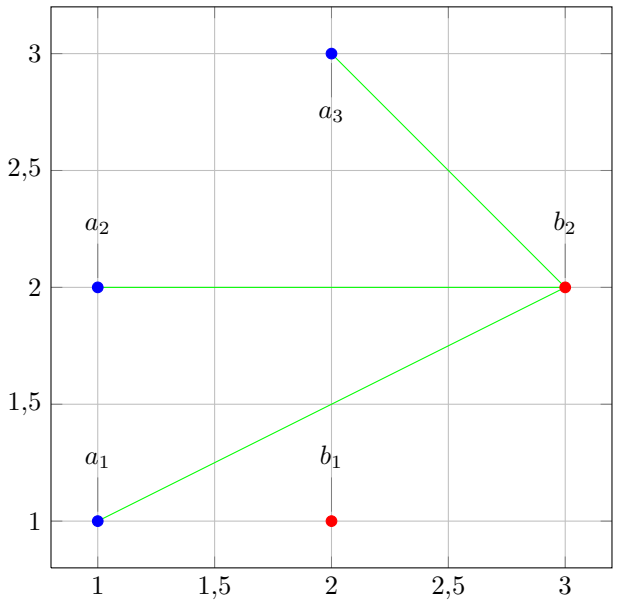
En la gráfica superior se toman las distancias $||a_1 - b_1||$ y $||a_1 - b_2||$ siendo

$||\cdot||$ la norma euclidiana. Obteniendo $d_{a_1,b_1} = 1$ y $d_{a_1,b_2} = 2{,}23$, por lo que para a_1 nos quedaríamos con la distancia d_{a_1,b_1} por ser la menor de ambas.

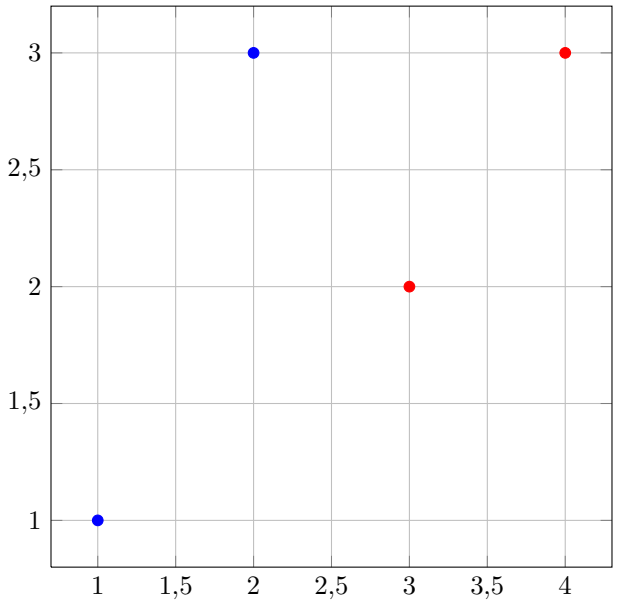


Se realiza el mismo proceso con a_2 y a_3 obteniendo que las menores distancias de éstos puntos con B son $d_{a_2,b_1} = 1{,}41$ y $d_{a_3,b_2} = 1{,}41$. De entre las tres distancias d_{a_1,b_1}, d_{a_2,b_1} y d_{a_3,b_2} tendremos que elegir la mayor, pero como hay igualdad entre las dos últimas se puede elegir cualquiera de ellas.





A continuación se ha operado $d_{b_q,a_p} = h(B,A)$ y se ha obtenido que las menores distancias son $d_{b_1,a_1} = 1$ y $d_{b_2,a_3} = 1{,}41$, de forma que el máximo será d_{b_2,a_3}. Dado que en este caso los valores obtenidos por $h(A,B)$ y $h(B,A)$ son iguales, podemos decir que la distancia Hausdorff entre los conjuntos A y B, $d_H(A,B) = 1{,}41$. En este ejemplo $h(A,B)$ y $h(B,A)$ han devuelto el mismo resultado, pero eso no es necesariamente cierto en todos los casos, como en el siguiente ejemplo:



En el primer ejemplo puede apreciarse donde pueden surgir los cuellos de botella a la hora de calcular una distancia entre dos conjuntos de puntos. El ejemplo tiene un grupo A con 3 elementos y un grupo B con 2. Para obtener la distancia han tenido que realizarse $3 \times 2 = 6$ medidas de distancia; en este caso siendo la norma euclidiana dos restas, dos potencias, una raíz y una comparación, cada distancia requiere un total de 6 operaciones. Tras la obtención de la distancia mínima entre un punto y el conjunto contrario, debe compararse con la anterior, excepto en el primer caso, y finalmente se hace una última comparación para obtener la distancia final. Esto hace que hayan hecho falta $(6 \times 6) + 2 + 1 + 1 = 40$ operaciones en para obtener la distancia. De aquí se deduce que el número de operaciones necesarias para calcular la distancia Hausdorff entre dos grupos de n y m elementos será $(n \times m \times 6) + (n + m - 1)$, por lo que se estima la complejidad computacional en $O(6n \cdot m)$.

4.2. Algoritmo

Según lo explicado en el apartado anterior resulta inmediato deducir la aplicación de la distancia Hausdorff a la detección de imágenes y formas en imágenes, ya que a fin de cuentas una imagen no es más que una agrupación de píxeles ordenados de una determinada forma, por lo tanto es posible definir un umbral para el valor de la distancia entre dos conjuntos de píxeles que determinen si las imágenes a comparar son la misma. Es más, es posible comparar conjuntos de píxeles que tienen un número de elementos deferentes y que la información que obtenemos tenga la misma precisión que en otro caso.

Por ejemplo haciendo uso de la distancia Hausdorff se pueden tomar unicamente los vértices de una figura y compararla consigo misma, pero teniendo menos puntos (a causa del ruido o de alguna ocultación en la imagen) y reconocer esta de forma precisa. Como la detección de vértices es una tarea costosa, en este trabajo se ha tomado la decisión de comparar contornos para realizar la detección.

El algoritmo que se presenta a continuación únicamente calcula la distancia de Hausdorff entre dos grupos de píxeles pertenecientes a los contornos de las imágenes que se quieren comparar, pero la decisión sobre si son o no la misma imagen debe tomarla el usuario. La búsqueda de un determinado elemento dentro de una imagen o la automatización de la decisión en base a la distancia obtenida son tareas posibles pero que quedan fuera del ámbito de este proyecto y que podrán realizarse en trabajos futuros.

Algorithm 1: Distancia Hausdorff

Data: A = bitmap1($m \times n$), B = bitmap2($p \times q$).
Result: Distancia d_H entre A y B.
begin

```
1  A ⟵ leer(bitmap1)
2  B ⟵ leer(bitmap2)
3  A' ⟵ detectarBordes(A)
4  B' ⟵ detectarBordes(B)
5  d_{A,B} ⟵ −1
6  d_{B,A} ⟵ −1
7  d_{temp} ⟵ −1
8  d_H ⟵ 0
9  for a ∈ A' do
10     if color(a) = blanco then
11         A'_c ⟵ coordenadasXY(a)
12 for b ∈ B' do
13     if color(b) = blanco then
14         B'_c ⟵ coordenadasXY(b)
15 for a' ∈ A'_c do
16     for b' ∈ B'_c do
17         if d_{temp} = −1 then
18             d_{temp} ⟵ ||a' − b'||
19         d_{temp} ⟵ min(d_{temp}, ||a' − b'||)
20     d_{A,B} ⟵ max(d_{temp}, d_{A,B})
21 d_{temp} ⟵ −1
22 for b' ∈ B'_c do
23     for a' ∈ A'_c do
24         if d_{temp} = −1 then
25             d_{temp} ⟵ ||b' − a'||
26         d_{temp} ⟵ min(d_{temp}, ||b' − a'||)
27     d_{B,A} ⟵ max(d_{temp}, d_{B,A})
28 d_H ⟵ max(d_{A,B}, d_{B,A})
```

En el algoritmo se distinguen dos partes, la primera que abarca de las líneas 1 hasta la 14 es la encargada de preparar y optimizar los datos para la segunda, que consiste en el resto del pseudocódigo.

Las líneas 1 a 4 indican métodos que se podrían haber descrito de forma

profunda en el algoritmo. Esto no se ha hecho porque estas operaciones no se han implementado como parte de este trabajo; en lugar de esto se han aprovechado las bibliotecas de OpenCV, puesto que el objetivo de este trabajo es la implementación eficiente de la distancia Hausdorff para la detección de imágenes, y quedaba fuera del ámbito la programación de éstos métodos y, además, las OpenCV ofrecen ya una programación optimizada y testada para la captura y tratamiento de imágenes que facilitan el punto de partida de este proyecto. Como su nombre en el pseudocódigo indica, estas funciones se encargan de leer las imágenes y realizar la detección de contornos sobre ellas. Las siguientes cuatro líneas, de la 5 a la 8, indican una inicialización de valores que resultan necesarios para el correcto funcionamiento del algoritmo. Este es un elemento más cercano a la codificación que de un pseudocódigo de alto nivel, pero se ha considerado incluirlo por claridad.

Los bucles 9-11 y 12-14 son una medida de optimización, y se encargan de guardar únicamente las posiciones de los píxeles que forman los contornos en las imágenes, de forma que el número de elementos con los que trabajar posteriormente quede muy reducido. La motivación es que en este algoritmo el resto de información de la imagen es prescindible, por tanto solo trabajaremos con aquella que es necesaria para el cálculo de la distancia de Hausdorff. Es en estos bucles donde pueden multiplicarse los valores de las coordenadas de uno de los contornos por una constante previamente calculada para conseguir tener invarianza rotacional y de escalado. No se muestra en el algoritmo al no formar parte de la idea original y por añadir el problema extra del cálculo del valor de la constante.

El resto del algoritmo presenta una forma en que pueden calcularse los valores de $h(A, B)$ y $h(B, A)$ sin necesidad de almacenar vectores o matrices con los rankings de distancias, ya que la comparación se realiza en cada iteración. La última línea es la comparación final para determinar el valor de la distancia Hausdorff entre los contornos de los dos mapas de bits originales.

Capítulo 5

Implementación

Este capítulo recoge la selección de tecnologías con las que se ha realizado el desarrollo y las razones de tales decisiones, así como una descripción del proceso de diseño del programa y la construcción de éste en sus diferentes variantes.

5.1. Tecnologías usadas

Según los requisitos expresados en el capítulo del estado del arte, se ha decidido utilizar C/C++ como lenguaje base para la programación del algoritmo. Esta decisión viene motivada por diversos motivos:

- Es un lenguaje abierto que permite la implementación de programas siguiendo prácticamente cualquier paradigma de programación, de forma que el programador no está restringido a diseñar la solución de una forma concreta por limitaciones en la herramienta.
- C/C++ es un lenguaje compilado, no interpretado, por lo que el rendimiento es mayor y ofrece un amplio marco de opciones de optimización mediante instrucciones al compilador, sin necesidad de cambiar el código del programa.
- Es portable, en el sentido de que se puede compilar para funcionar en la plataforma deseada.
- Todas las herramientas de paralelización están disponibles para este lenguaje, de forma que pueden realizarse diferentes optimizaciones y comparaciones de rendimiento sin necesidad de tener que reescribir el programa.

Como herramienta de paralelización se ha decidido usar OpenMP porque ofrece una interfaz sencilla que permite realizar la paralelización introduciendo pocos cambios sobre la implementación secuencial. Esta herramienta es independiente de la máquina en que se ejecute y se encuentran disponible de forma gratuita para la plataforma C/C++.

Para la solución del problema es necesario hacer uso de una herramienta de apoyo que nos permita leer y manipular imágenes. En este caso se ha elegido OpenCV, puesto que son unas bibliotecas basadas en C/C++, probadas, con amplia documentación y, en su mayor parte, optimizadas, que nos permiten

realizar con sencillez las operaciones que son necesarias para el proyecto. Otra característica interesante es que al ser de código abierto, es posible acceder a las funciones y realizar optimizaciones o modificaciones que puedan ser deseables o necesarias, aunque esto no entraría dentro del ámbito del presente trabajo, deja una puerta abierta para modificaciones futuras.

5.2. Diseño

Partiendo del algoritmo mostrado en la sección anterior se diseña el siguiente diagrama de clases:

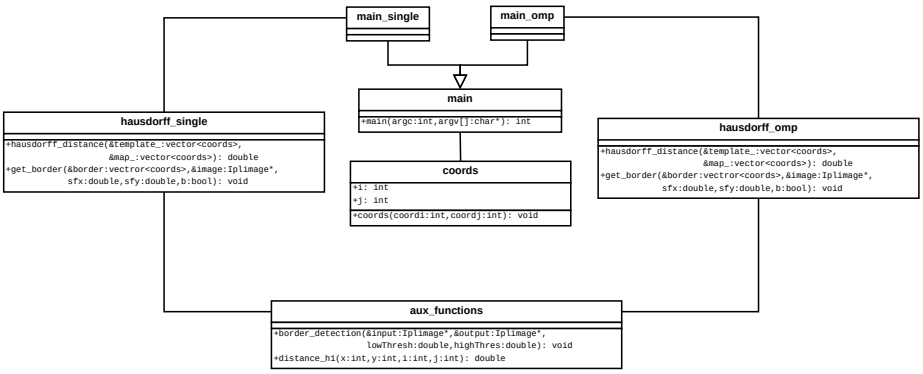


Figura 5.1: Diagrama de clases del programa.

Puede verse que la característica principal del diseño consiste en la modularidad, de forma que puedan realizarse diferentes implementaciones del calculo de la distancia de Hausdorff en base a la maquina o necesidades de uso. En la solución propuesta solo se presentan una implementación secuencial optimizada y su versión mejorada con OpenMP. Debido a la poca cantidad de elementos del programa, no ha sido necesario aplicar ningún patrón para resolver el diseño, pero en caso de realizarse ampliaciones o se conectase a un sistema más complejo si podría ser necesario hacer uso de estos para facilitar dicha tarea.

Detalle de la funcionalidad ofrecida por cada clase:

- **main:** Contiene la función de entrada al programa donde se inicializan las variables y llama al resto de clases y funciones. Se utiliza como clase abstracta para que hereden las implementaciones concretas del algoritmo.
- **main_single:** Contiene las variables y llamadas especificas para la ejecución del algoritmo de forma secuencial.
- **main_omp:** Contiene las variables y llamadas que implementan la versión paralelizada con OpenMP.
- **coords:** Estructura que permite almacenar las coordenadas de un píxel de un mapa de bits.
- **hausdorff_single:** Implementación secuencial del algoritmo para el cálculo de la distancia de hausdorff.

- **hausdorff_omp:** Modificación del algoritmo secuencial para que se ejecute de forma paralela y optimizada con OpenMP.
- **aux_functions:** Funciones auxiliares: Cálculo de la norma euclidea y deteción de bordes. Podrían incluirse otras normas o funciones más optimizadas.

La decisión de separar los programas principales por implementación es por un motivo de rendimiento. Es posible llamar a las funciones de calculo para la versión secuencial y para la versión paralela desde la misma función main, pero en ese caso es posible que la que se llamase después aprovechase la caché de la anterior y eso falsearía las medidas de tiempos. Este problema no ocurre si cada versión tiene su propio ejecutable. La creación de la estructura *coords* ha estado motivada por el número de píxeles a comparar entre dos imágenes; supongamos una comparación de una imagen consigo misma de $500x500$, eso serían 250000 píxeles, dato que conllevaría 250000^2 comparaciones. Por tanto se ha decidido, una vez detectados los contornos que se encuentran en las imágenes, recorrer estas y guardar la posición de los píxeles que forman el contorno, de manera que en la comparación posterior solo se usen estos datos, en lugar de toda la imagen, disminuyendo de manera considerable la cantidad de cálculos que deben realizarse.

5.3. Secuencial

La implementación básica realiza una aproximación secuencial del algoritmo expuesto en la sección correspondiente. Esto puede apreciarse especialmente en el *main* de esta versión, ya que sigue punto por punto el algoritmo:

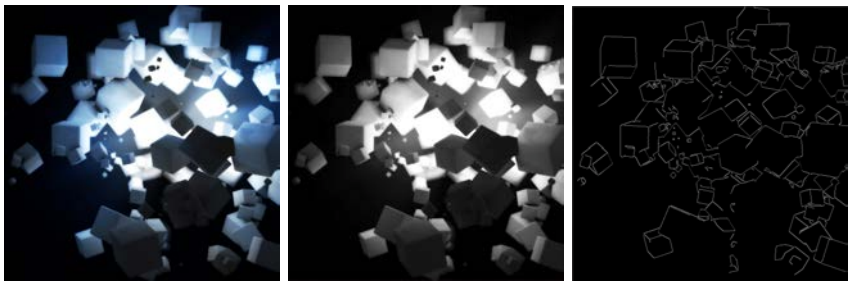
```
int main(int argc, char* argv[]){
  vector<coords> template_, map_;
  hausdorff_single haus_sing;
  double dist, threshold = -1, t1, t2,t3,t4,
    sfx = 0,sfy = 0,sfx_max,sfx_min,sfy_max,sfy_min;
  IplImage *input, *map, *output_template, *output_map;
  int x;
  timeval tim;

  while ((x = getopt (argc, argv, "t:m:")) != -1){
    switch (x)     {
      case 't':
        input = cvLoadImage( optarg , 0);
        break;
      case 'm':
        map = cvLoadImage( optarg , 0);
        break;
      case '?':
        if((optopt == 't') || (optopt == 'm')){
          cout << "El formato de entrada es:" << endl;
          cout << "./main_<nombre_version> -t <template_path>
            -m <map_path>" << endl;
        }
        else{
```

```
            cout << "El formato de entrada es:" << endl;
            cout << "./main_<nombre_version> -t <template_path>
              -m <map_path>" << endl;
          }
          return 1;
        default:
          cout << "El formato de entrada es:" << endl;
          cout << "./main_<nombre_version> -t <template_path>
            -m <map_path>" << endl;
          return 1;
      }
    }

    // Set up images
    output_template = cvCreateImage(cvGetSize(input),IPL_DEPTH_8U, 1);
    output_map = cvCreateImage(cvGetSize(map),IPL_DEPTH_8U, 1);

    // Border detection
    border_detection(input,output_template,200,400);
    border_detection(map, output_map,200,400);

    // Read border
    haus_sing.get_border(template_,output_template,sfx,sfy,false);
    haus_sing.get_border(map_,output_map,sfx,sfy,false);

    //Single
    gettimeofday(&tim,NULL);
    t1=tim.tv_sec+(tim.tv_usec/1000000.0);

    //Hausdorff distance measurement
    dist = haus_sing.hausdorff_distance(template_,map_);

    gettimeofday(&tim,NULL);
    t2=tim.tv_sec+(tim.tv_usec/1000000.0);

    // Output
    cout << "T: " << t2-t1 << " D: " << dist << " PElm1: "
      << template_.size() << " PElm2: " << map_.size() << endl;

    cvReleaseImage( &input );
    cvReleaseImage( &output_template );
    cvReleaseImage( &output_map );

    return 0;
}
```

Listing 5.1: main_single()

Dejando de lado la inicialización de parámetros, vemos que las imágenes se cargan durante la gestión de argumentos. Esta gestión se ha implementado usando la funcion *getopt()* porque permite añadir más parámetros de forma sencilla y flexible. A continuación se ve que el programa primeramente transforma los mapas de bits en a un formato que permita su manipulación desde OpenCV, para luego poder aprovechar la funcionalidad de esta biblioteca para detectar

los bordes de las imágenes de entrada. Para facilitar el proceso de detección de contornos las imágenes se han convertido a escala de grises. En este punto la ejecución se encontraría en esta situación:



(a) Original (b) Escala de grises (c) Contornos

Figura 5.2: Ejemplo de manipulación de la imágen durante la ejecución del programa.

La calidad de la detección de contornos varía dependiendo de dos factores, la versión de OpenCV que se utilice, ya que las más modernas tendrán mejores mecanismos de detección, y los dos últimos parámetros de la función *border_detection()*.

```
void border_detection(IplImage* &input, IplImage* &output,
    double lowThresh, double highThresh){
  int N = 7;

  IplImage* img = input;
  IplImage* img_b = cvCreateImage( cvSize(img->width+N-1
    ,img->height+N-1), img->depth, img->nChannels );
  output = cvCreateImage( cvGetSize(img_b), IPL_DEPTH_8U, 1);

  // Add convolution boarders
  CvPoint offset = cvPoint((N-1)/2,(N-1)/2);
  cvCopyMakeBorder(img, img_b, offset,
    IPL_BORDER_REPLICATE, cvScalarAll(0));

  int aperature_size = N;

  cvCanny( img_b, output, lowThresh*N*N, highThresh*N*N, aperature_size );
}
```

Listing 5.2: border_detection()

En el ejemplo anterior los contornos se han detectado en la versión 2.3 de OpenCV en Windows, pero en la versión de las pruebas en el *cluster* se ha ejecutado bajo OpenCV 2.4.2 en Linux, lo cual produce discrepancias a favor de la versión ejecutada en las pruebas, ya que se detectan mejor los bordes con los mismos parámetros. Solo se ha mostrado una imagen, pero en la ejecución habría dos en memoria, la original y aquella con la que se quiere comparar.

La siguiente llamada es a la función *get_border*, que se encarga de guardar las coordenadas de los píxeles que forman el contorno detectado en la imagen

en una estructura de memoria *vector<coords>*.

```
void hausdorff_single::get_border(vector<coords> &border,
    IplImage* &image, double sfx, double sfy, bool b){
  int x = 0,y = 0,aux_i = 0,aux_j = 0;

  for (int i = 0; i < image->height; i++){
    for (int j = 0; j < image->width; j++){
      CvScalar scv = cvGet2D(image,i,j);
      if (scv.val[0] == 255)      {
        if(aux_i == 0&& aux_j == 0){
          aux_i = i;
          aux_j = j;
        }
        if(aux_i != i){
          y += (i-aux_i);
          aux_i = i;
        }
        if(aux_j != j){
          x += (j-aux_j);
          aux_j = j;
        }
        border.push_back(coords(y,x));
      }
    }
  }
}
```

Listing 5.3: get_border()

```
coords::coords(int coordi, int coordj){
  i = coordi;
  j = coordj;
}
```

Listing 5.4: Estructura coords

Se ha decidido usar un tipo *vector* dado que a priori es imposible conocer la cantidad exacta de píxeles que se van a guardar para poder crear una *array*. La función sencillamente recorrerá el mapa de bits y almacenará en el *vector<T>* las coordenadas de los píxeles que sean blancos. Las coordenadas que almacena no serán la posición del píxel en la imagen original, sino que irá numerándolos a medida que los encuentra, de esta forma el algoritmo será invariante ante traslaciones, ya que la posición del contorno en la imagen deja de importar. Esta modificación tiene en cuenta posibles saltos y huecos de forma que no se pierda la proporción de los contornos.

A continuación se inician las medidas del tiempo para calcular de forma exacta la tardanza del cálculo de la distancia Hausdorff y se ejecuta el método principal del programa.

```
double hausdorff_single::hausdorff_distance(vector<coords> &template_,
    vector<coords> &map_){
  double distances = -1, temp = 0, distance_A = -1, distance_B = -1;
  int  tsize , msize;
   tsize  = template_.size();
  msize = map_.size();

   // Cálculo de la  distancia  del  conjunto A al B

  for(int  t  = 0; t<tsize;t++){
    distances  = distance_h1(template_[t].i,template_[t].j,map_[0].i,map_[0].j);
    for(int  k  = 1; k<msize;k++){
      temp = distance_h1(template_[t].i,template_[t].j,map_[k].i,map_[k].j);
      distances  = std::min(distances,temp);
    }
    distance_A = std::max(distances,distance_A);
  }

   // Cálculo de la  distancia  del  conjunto B al A

  for(int  t  = 0; t<msize;t++){
    distances  = distance_h1(map_[t].i,map_[t].j,template_[0].i,template_[0].j);
    for(int  k  = 1; k<tsize;k++) {
      temp = distance_h1(map_[t].i,map_[t].j,template_[k].i,template_[k].j);
      distances  = std::min(distances,temp);
    }
    distance_B = std::max(distances,distance_B);
  }

  return std::max(distance_A,distance_B);
}
```

Listing 5.5: hausdorff_distance()

La función consiste en dos bucles anidados, el primero recorre los píxeles de los contornos de la primera imagen, el segundo, para cada píxel, recorre todos los píxeles de los contornos de la segunda imagen y calculará las distancias entre estos y se guardará siempre la menor. Al finalizar el recorrido para un punto dado de la primera imagen seleccionará aquella distancia que sea mayor entre la nueva y el máximo anterior. Este proceso se realiza en el sentido imagen1 $\rightarrow$ imagen2 y luego en el sentido imagen1 $\leftarrow$ imagen2. Finalmente devolverá el mayor valor de entre ambas. Las variables *tsize* y *msize* se han creado para evitar tener que llamar a la función *size()* de *vector<coords>* en cada iteración de los bucles y evitar así perder algo de tiempo.

La medida de la distancia será la norma euclidiana, aunque podrían usarse otras:

```
double distance_h1(int x, int y, int i,  int  j)
{
  return sqrt(pow((x-i),2)+pow((y-j),2));
}
```

Listing 5.6: distance_h1()

Completando el flujo del *main_single()*, los siguientes pásos en la ejecución son la finalización de la medida del tiempo tras la obtención de la distancia, la impresión de los resultados por pantalla: El tiempo, la distancia y los píxeles de los contornos de cada imagen. Por último se limpia la memoria y termina el programa.

5.4. Paralelización 1

Optimizar el código anterior con OpenMP resulta casi inmediato. Se ha adaptado el *main_single()* en el *main_omp()* para hacer las llamadas a la clase *hausdorff_omp* en lugar de la clase *hausdorff_single* y se han modificado los nombres de algunas variables para reflejar que se trata de una versión diferente. Los cambios más importantes se han producido en la función *hausdorff_distance()*:

```
double hausdorff_omp::hausdorff_distance(vector<coords> &template_,
    vector<coords> &map_){
  int nthreads, tsize, msize;
  double *distA, *distB, distance_A = -1, distance_B = -1;
  tsize = template_.size();
  msize = map_.size();

  #pragma omp parallel{
    #pragma omp single{
      nthreads = omp_get_num_threads();
      distA = new double[nthreads];
      distB = new double[nthreads];
      for(int i = 0; i < nthreads;i++){
        distA[i] = -1;
        distB[i] = -1;
      }
    }
  }

   // Cálculo de la distancia del conjunto A al B

  #pragma omp parallel for schedule(static)
  for(int t = 0; t<tsize;t++){
  int tid = omp_get_thread_num();
  double temp = 0, dist = -1, aux;

    dist = distance_h1(template_[t].i,template_[t].j,map_[0].i,map_[0].j);
    for(int k = 1; k<msize;k++){
      temp = distance_h1(template_[t].i,template_[t].j,map_[k].i,map_[k].j);
      dist= std::min(dist,temp);
    }
    distA[tid] = std::max(dist,distA[tid]);
    aux = std::max(dist,aux);
  }

   // Cálculo de la distancia del conjunto B al A

  #pragma omp parallel for schedule(static)
```

```
  for(int t = 0; t<msize;t++){
  int tid = omp_get_thread_num();
  double temp = 0, dist = -1, aux;

    dist = distance_h1(map_[t].i,map_[t].j,template_[0].i,template_[0].j);
    for(int k = 1; k<tsize;k++){
      temp = distance_h1(map_[t].i,map_[t].j,template_[k].i,template_[k].j);
      dist = std::min(dist,temp);
    }
    distB[tid] = std::max(dist,distB[tid]);
    aux = std::max(dist,aux);
  }

  for(int i = 0; i < nthreads;i++) distance_A = std::max(distA[i],distance_A);
  for(int i = 0; i < nthreads; i++) distance_B = std::max(distB[i],distance_B);

  delete distA;
  delete distB;

  return std::max(distance_A,distance_B);
}
```

Listing 5.7: hausdorff_distance()

Para que la paralelización sea correcta y cada hilo de ejecución tenga privacidad en sus variables, de forma que no se alcancen condiciones de carrera o se sobreescriban valores que no deberían, se han añadido las siguientes variables:

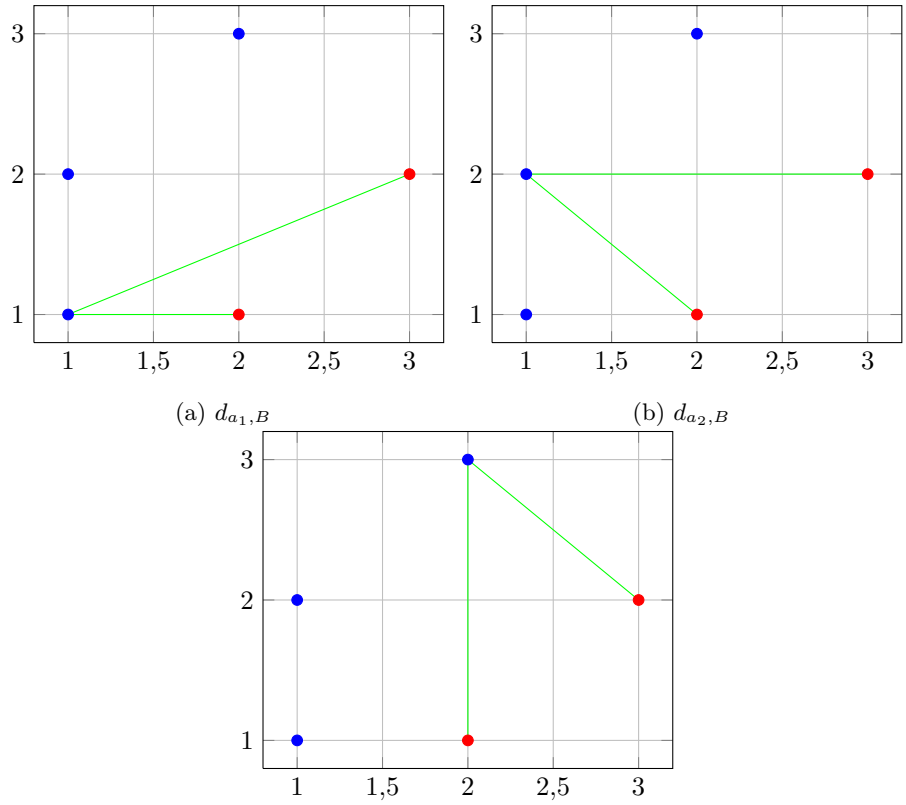
- **double *distA y *distB:** Cada una de estas será un *array* con tantas posiciones como hilos de ejecución existan. Es necesario crear estas variables para que cada hilo calcule sus distancias de forma independiente a los demás. Al finalizar el bucle se recorrerán estos *arrays* para seleccionar los valores deseados.
- **int nthreads:** La variable *nthreads* guardará la información de cuántos threads están ejecutando el programa, de forma que no tenga que llamarse a la función *omp_get_num_threads()* repetidas veces.

El primer fragmento de código rodeado por la directiva *#pragma omp parallel* se utiliza para averiguar los hilos existentes e inicializar los *arrays* distA y distB. Esto debe hacerse así ya que fuera de una sección paralela no es posible averiguar el número de hilos disponibles.

A continuación se han incluido las directivas *#pragma omp parallel for schedule(static)* delante de cada bucle principal, de forma que serán éstos los que repartan las tareas entre los hilos disponibles. Esta indicación hace que el compilador modifique los valores de inicio y fin del bucle para cada hilo, de forma que se reparten las tareas equitativamente, y si hay un total de m elementos a recorrer en el bucle y n hilos de ejecución disponibles, cada hilo realizará $\frac{m}{n}$ tareas. Ésto es así porque se indica como planificación -*schedule*- estática. OpenMP permite indicar una planificación dinámica, en la que se asigna una tarea a cada hilo y, al terminar de ejecutarla, se asignan más tareas mientras queden. Desde un principio se ha decidido no utilizar esta aproximación a la paralelización ya que no resulta optimo, dado que el cálculo de las distancias

para cada punto se realiza muy rápido, se pierde más tiempo asignando tareas que el tiempo actual de procesamiento de las distancias.

La idea de esta optimización consiste en dividir el cálculo de distancias de cada punto de una imagen a la otra, de forma que si el contorno de la primera contiene m píxeles, cada hilo calcule las distancias de $\frac{m}{n}$ píxeles de la primera imagen a la segunda. Explicado de forma gráfica la versión secuencial calcula las distancias de esta forma:



(a) $d_{a_1,B}$ (b) $d_{a_2,B}$

(c) $d_{a_3,B}$

Figura 5.3: Ejecución secuencial del cálculo de las distancias del grupo A al B

Frente a la versión paralela:

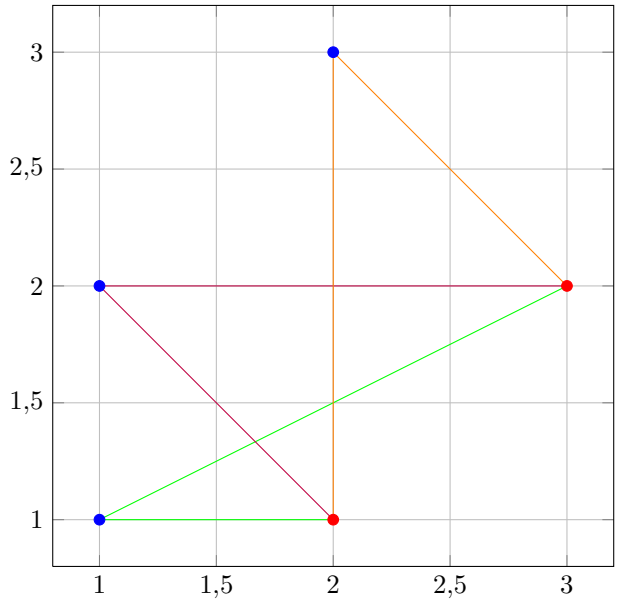


Figura 5.4: $d_{A,B}$ paralelizado

En la segunda figura se calculan de forma simultanea $d_{a_1,B}$, $d_{a_2,B}$ y $d_{a_3,B}$, en lugar de tener que esperar a que haya terminado el cálculo anterior antes de pasar al siguiente.

Para finalizar la ejecución, se compararán las distancias obtenidas por cada hilo y se seleccionarán las que interesan, devolviendo finalmente el valor de la distancia de Hausdorff entre ambas imágenes.

Parte III

Resultados y conclusiones

Esta última sección presenta la experimentación realizada: La batería de pruebas, entorno de ejecución de estas, resultados obtenidos y las conclusiones que se extraen.

Capítulo 6

Resultados

6.1. Batería de Pruebas

La batería de pruebas se compone de dos grupos de imágenes, uno de ellos servirá para comprobar como evolucionan los tiempos de ejecución y la medida de la distancia comparando las mismas imágenes a diferentes resoluciones y el segundo es solo una medida de como cambia el rendimiento aumentando la complejidad de la imagen manteniendo la misma resolución en todo el grupo.

El primer grupo consistirá de 20 imágenes, de las cuales 10 tendrán una A mayúscula sobre un fondo blanco y las otras 10 serán iguales salvo por un defecto en la tipografía. Las 10 parejas serán por tanto como se muestra a continuación:

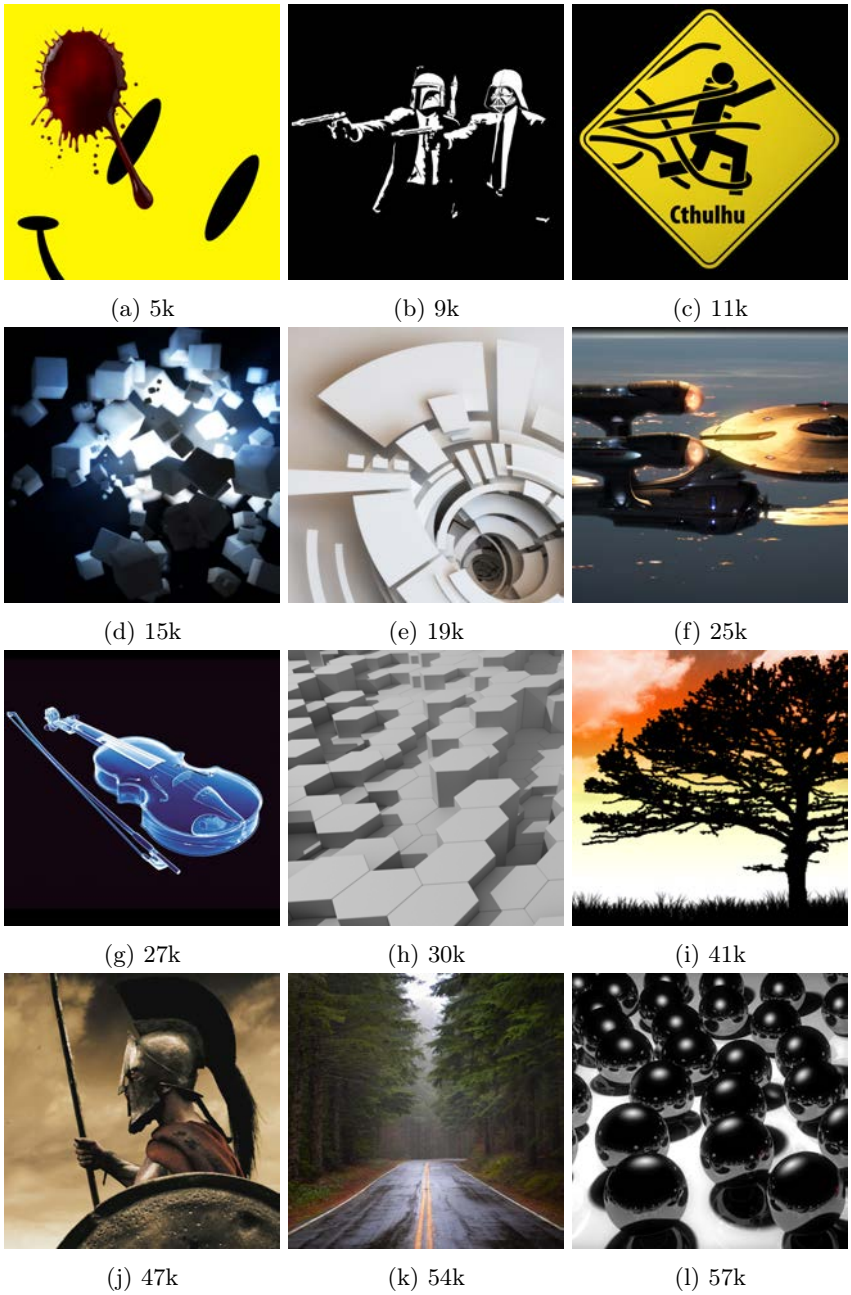
(a) Letra normal. (b) Letra con 'agujero'.

Figura 6.1: Patrón del primer grupo de pruebas.

Las resoluciones aumentarán desde 50x50 píxeles hasta 1125x1125 píxeles. Con este conjunto de imágenes se pretende determinar la variación de la distancia de Hausdorff según mejora la resolución en una misma pareja de imágenes.

El segundo grupo tendrá 15 imágenes de 1000x1000 píxeles cada una, en las que el número de contornos y su complejidad aumenta progresivamente. El

conjunto contendrá las siguientes imágenes:



(a) 5k (b) 9k (c) 11k

(d) 15k (e) 19k (f) 25k

(g) 27k (h) 30k (i) 41k

(j) 47k (k) 54k (l) 57k

Figura 6.2: Conjunto de imágenes del segundo grupo de pruebas. El nombre indica de forma aproximada el número de píxeles de los contornos detectados en el estudio de selección.

(a) 68k (b) 84k (c) 108k

Figura 6.3: Conjunto de imágenes del segundo grupo de pruebas. El nombre indica de forma aproximada el número de píxeles de los contornos detectados en el estudio de selección.

Con este segundo grupo de imágenes se va a estudiar la variación de los tiempos en el cálculo de la distancia de Hausdorff al aumentar la complejidad interna de la imagen objetivo. Esto es, cómo cada imagen contiene más contornos que la previa, para obtener una medida estable de los tiempos se compara consigo misma, evitando así hacer estudios de contornos con cantidades de píxeles diferentes.

6.2. Medidas tomadas

Para el desarrollo experimental se han tomado diez medidas de los tiempos de ejecución y se ha realizado la media para amortiguar las desviaciones que puedan producirse debido a procesos internos de la máquina de pruebas u otras causas.

La máquina sobre la que se han ejecutado estas pruebas posee las siguientes características:

- **Tipo de CPU:** Intel Westmere EX.
- **Reloj de la CPU:** 1.87 Ghz.
- **Sockets:** 4.
- **Núcelos por Socket:** 6.
- **Hilos por Núcelo:** 2.
- **Total Hilos de ejecución disponibles:** $2 \cdot 6 \cdot 4 = 48$.
- **Niveles de Caché:** 3.
- **Tamaño por nivel de Caché:** Nivel 1: 32 kb. Nivel 2: 256kb. Nivel 3: 18MB.
- **Sistema Operativo:** Ubuntu 10.10.

Con este equipo se han obtenido los siguientes datos:

1. Distancias de Hausdorff para el primer conjunto de imágenes.

2. Tiempos de ejecución para la implementación secuencial con los conjuntos de prueba.

3. Tiempos de ejecución de OpenMP con 2, 4, 8, 12, 16, 24 y 48 hilos de ejecución para las dos baterias de imágenes.

El número máximo de núcleos en la maquina de pruebas es 24, por lo que se debe destacar que los 48 hilos de ejecución no equivalen a 48 núcleos reales, es posible alcanzar este número de hilos gracias a la tecnología *HyperThreading* que llevan los procesadores Intel en que se han realizado las pruebas.

A continuación se muestran las tablas con los resultados del primer conjunto de prueba:

Res(px)	d_H	Tiempo (s) por Núcleos						
		1	2	4	8	12	24	48
50^2	6	0,00164	0,00103	0,00073	0,00059	0,00071	0,00186	0,11763
125^2	19	0,00843	0,00481	0,00244	0,00148	0,00127	0,00199	0,11223
250^2	37	0,0303	0,015767	0,00798	0,00458	0,00332	0,00367	0,13534
375^2	56	0,06395	0,03165	0,01617	0,00838	0,00583	0,00623	0,10992
500^2	75	0,11638	0,057087	0,02915	0,01561	0,01015	0,00988	0,12756
625^2	91	0,17567	0,08879	0,04450	0,02312	0,01545	0,01494	0,14609
750^2	112	0,25123	0,12582	0,06286	0,03181	0,02153	0,0203	0,13951
875^2	132	0,33923	0,16731	0,08435	0,04259	0,02887	0,02711	0,13899
1000^2	151	0,43753	0,21947	0,11126	0,05621	0,03797	0,03493	0,12521
1125^2	167	0,54934	0,27743	0,13921	0,06989	0,047	0,04535	0,13911

Tabla 6.1: Resultados del primer conjunto de pruebas

y sus desviaciones:

Res(px)	Desviación $\cdot 10^{-5}$(s)						
	1	2	4	8	12	24	48
50^2	±18	±2	±2	±2	±3	±26	±2538
125^2	±3	±69	±2	±4	±8	±22	±3851
250^2	±42	±31	±2	±46	±14	±44	±6463
375^2	±167	±16	±28	±12	±8	±63	±3610
500^2	±316	±95	±59	±283	±15	±65	±1072
625^2	±404	±196	±44	±51	±16	±51	±1888
750^2	±713	±213	±99	±71	±45	±92	±5874
875^2	±961	±163	±80	±28	±19	±87	±1682
1000^2	±576	±85	±175	±96	±86	±90	±1869
1125^2	±1205	±573	±251	±59	±26	±714	±2703

Tabla 6.2: Desviaciones típicas del primer conjunto de pruebas

Puede apreciarse que se cumplen las dos tendencias previsibles para este caso: el aumento de d_h con la resolución, y la disminución de tiempos al aumentar el número de hilos. La excepción se encuentra en la ejecución con 48 hilos, se detallan las causas de este hecho en la sección de análisis.

Tablas con los tiempos del segundo conjunto de prueba:

Edge(px)	Tiempo (s)						
	1	2	4	8	12	24	48
7219	10,2737	5,22329	2,5748	1,29506	0,86467	0,70346	0,75935
11198	24,5515	12,0182	6,0755	3,02743	2,07227	1,55494	1,40662
15040	43,9736	22,0362	11,163	5,57305	3,70774	2,86345	2,3
20189	79,8143	9,68483	20,0381	9,95972	6,71667	5,0772	3,55759
19490	73,7684	37,2538	18,5248	9,28752	6,26683	4,21938	3,41355
25576	126,798	63,8514	31,4563	15,8205	10,4211	6,55512	5,34347
27927	154,715	76,4067	37,7171	18,873	12,6095	8,41379	6,38066
30337	178,451	88,8429	44,5424	22,2716	14,9378	9,15289	7,52146
41533	334,511	167,754	83,9912	41,6992	28,0701	17,1608	13,3931
47141	432,148	218,655	107,905	54,7996	35,6386	25,1396	17,2869
54895	581,111	290,324	145,304	73,1556	48,236	29,3764	23,1207
57726	648,47	323,514	62,5783	80,9053	54,059	31,0153	5,8528
68761	913,015	455,547	225,879	113,65	75,2916	46,3124	35,9077
83942	1364,87	676,966	40,135	170,125	113,143	69,1532	53,8177
10807	2257,29	1118,41	566,366	278,62	187,434	104,356	88,5741

Tabla 6.3: Resultados del segundo conjunto de pruebas

y sus desviaciones:

Edge(px)	Desviación (s)						
	1	2	4	8	12	24	48
7219	$\pm 0,23$	±0,13	±0,02	±0,02	±0,01	±0,03	±0,07
11198	$\pm 0,7$	±0,03	±0,05	±0,005	±0,04	±0,19	±0,22
15040	±0,96	±0,19	±0,28	±0,12	±0,06	±0,31	±0,26
20189	±2,31	±0,88	±0,44	±0,21	±0,16	±0,58	±0,22
19490	±1,11	±1,07	±0,31	±0,16	±0,15	±0,75	±0,45
25576	±3	±1,75	±0,57	±0,35	±0,003	±1,31	±0,28
27927	±11,25	±2,13	±0,35	±0,32	±0,28	±1,75	±0,33
30337	±3,88	±1,54	±0,77	±0,15	±0,27	±1,92	±0,39
41533	±6,7	±4,39	±2,07	±0,79	±0,81	±3,51	±0,22
47141	±10,7	±6,53	±2,37	±1,63	±0,12	±5,22	±0,32
54895	±9,7	±4,94	±2,53	±1,65	±0,34	±5,73	±0,44
57726	±14,74	±5,91	±4,68	±1,9	±1,41	±6,45	±0,68
68761	±17,43	±8,33	±0,48	±2	±0,54	±10,01	±0,63
83942	±33,82	±15,45	±6,52	±3,22	±2,08	±16,67	±1,33
108076	±48,32	±19,43	±12,99	±0,71	±4,27	±10,04	±2,01

Tabla 6.4: Desviaciones típicas del segundo conjunto de pruebas

Al igual que en las primeras pruebas, estos resultados son prometedores y se puede afirmar que la paralelización resulta extremadamente beneficiosa para el cálculo de la distancia de Hausdorff en el problema estudiado.

6.3. Análisis y Comparativa

Se muestran los resultados obtenidos en la sección anterior de forma gráfica para poder realizar el análisis sobre estos:

Gráfica del primer conjunto de prueba en el que se representan las distancias obtenidas frente a las resoluciones:

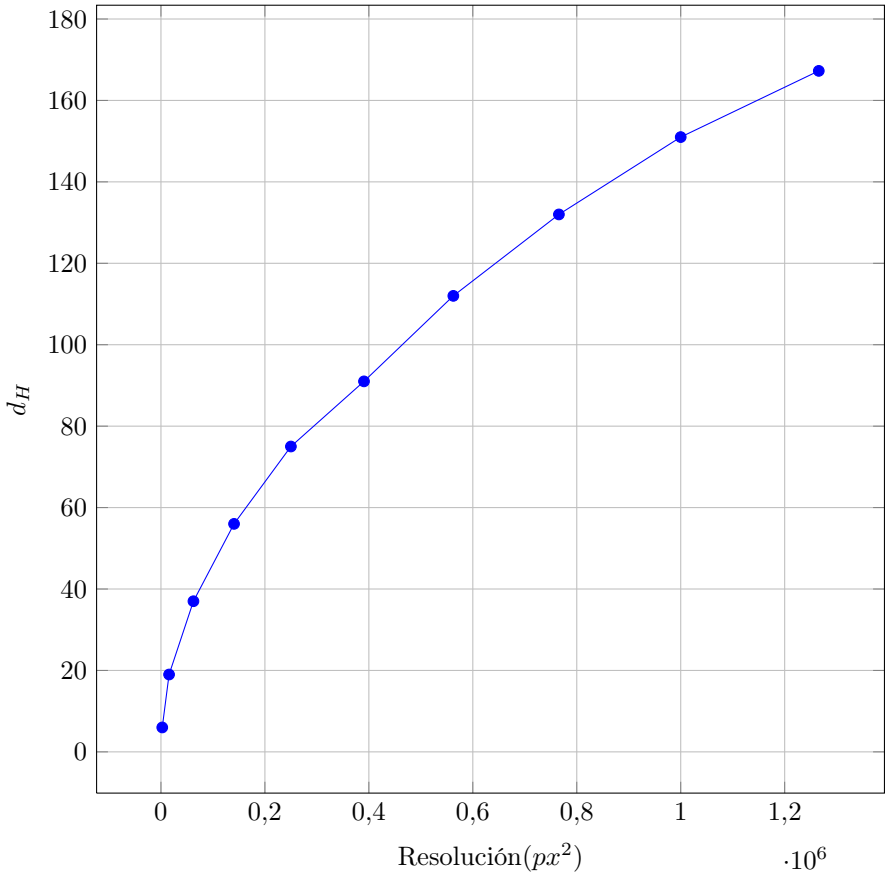


Figura 6.4: Distancia frente a Resolución

La figura 6.4 muestra la representación de las medidas de distancia obtenidas frente a la resolución de las imágenes.

Ajustando la curva con una función exponencial utilizando el programa de análisis matemático y estadístico OriginPro, se obtiene el siguiente ajuste y sus parámetros:

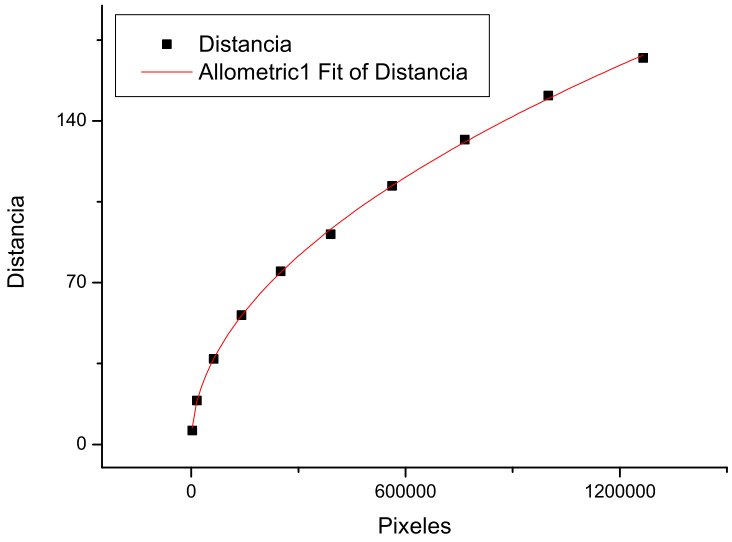


Figura 6.5: Ajuste

- Ecuación de ajuste: $y = a \cdot x^b$
- $a = 0{,}14 \pm 0{,}011$
- $b = 0{,}504 \pm 0{,}005$
- Coeficiente de regresión (r^2)= 0,9995

El buen valor del coeficiente de regresión 0.9995, confirma que el crecimiento de la distancia de Hausdorff con el aumento de la resolución sigue la ecuación siguiente:

$$d_H = 0{,}14 \cdot X^{0{,}504} \tag{6.3.0.1}$$

Donde d_H es el valor de la distancia de Hausdorff y X el número total de píxeles en la imagen.

Es necesario realizar más pruebas con un otros conjuntos de imágenes a distintas resoluciones para verificar que efectivamente la distancia se comporta de esta manera, pero este estudio se sale fuera del ámbito de este trabajo por lo que se propone como parte de un proyecto futuro.

Representación de tiempos de proceso frente a resoluciones para las medidas tomadas con diferente número de hilos de ejecución:

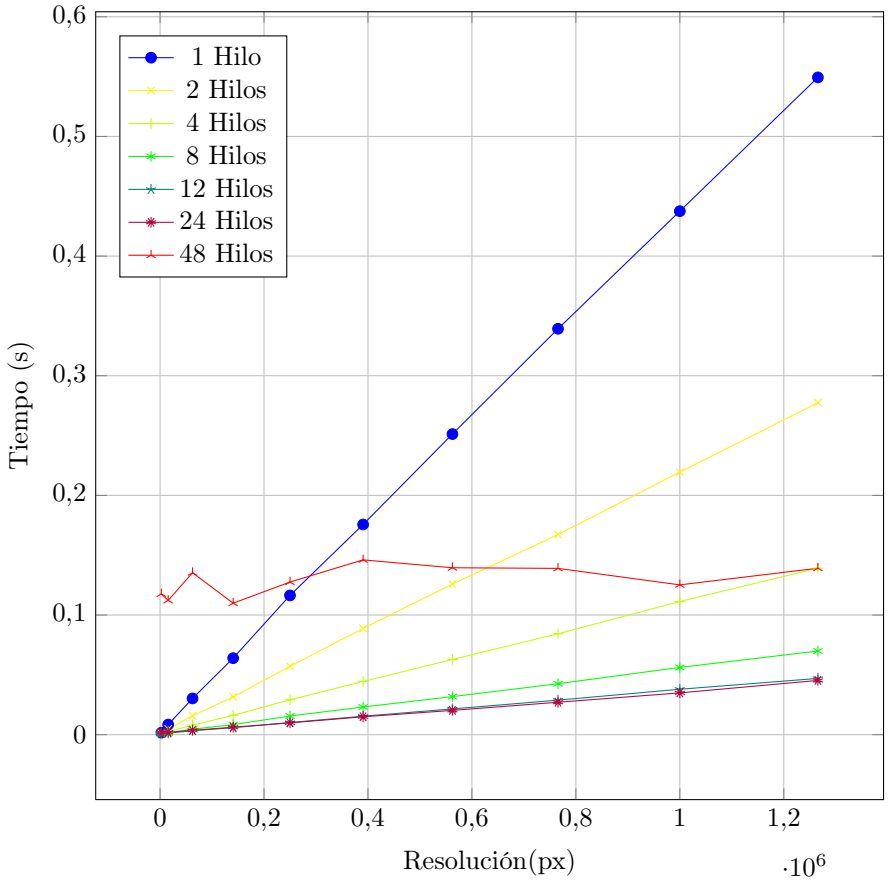
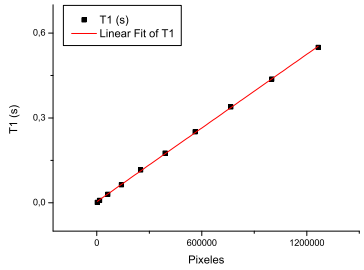


Figura 6.6: Tiempo frente a Resolución

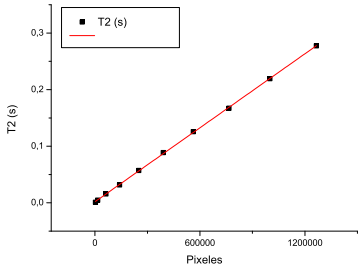
Haciendo los ajustes líneales que estos resultados sugieren, obtenemos:

Hilos	Ordenada en el origen	Pendiente	r^2
1	$0,00397 \pm 0,00139$	$4,3407 \cdot 10^{-7} \pm 2,27832 \cdot 10^{-9}$	0,99975
2	$0,00181 \pm 5,33706 \cdot 10^{-4}$	$2,17954 \cdot 10^{-7} \pm 8,72305 \cdot 10^{-10}$	0,99986
4	$0,00104 \pm 2,58797 \cdot 10^{-4}$	$1,09592 \cdot 10^{-7} \pm 4,22985 \cdot 10^{-10}$	0,99987
8	$0,00102 \pm 2,53937 \cdot 10^{-4}$	$5,47728 \cdot 10^{-8} \pm 4,15042 \cdot 10^{-10}$	0,99948
12	$8,39143 \cdot 10^{-4} \pm 1,14018 \cdot 10^{-4}$	$3,67459 \cdot 10^{-8} \pm 1,86355 \cdot 10^{-10}$	0,99977
24	$0,00146 \pm 1,91978 \cdot 10^{-4}$	$3,40507 \cdot 10^{-8} \pm 3,13774 \cdot 10^{-10}$	0,99924
48	$0,12267 \pm 0,00532$	$1,45602 \cdot 10^{-8} \pm 8,69396 \cdot 10^{-9}$	0,16704

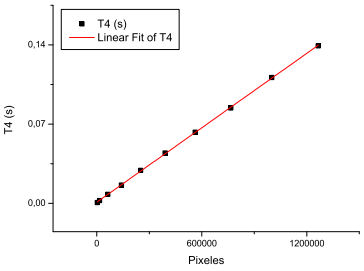
Tabla 6.5: Ajuste líneal de los tiempos de ejecución respecto a la resolución para cada grupo de Hilos.



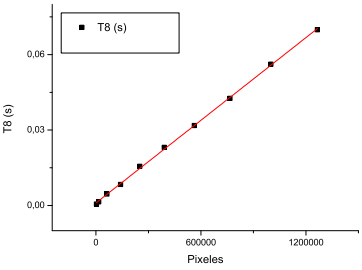
(a) Ajuste 1 Th.



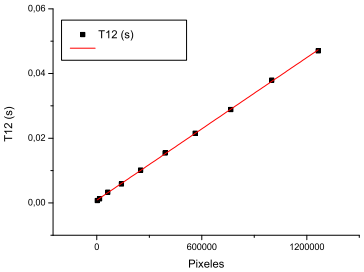
(b) Ajuste 2 Th.



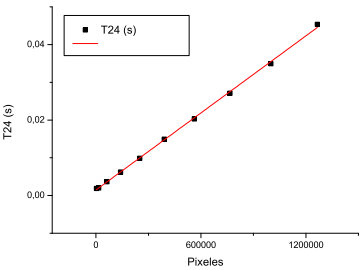
(c) Ajuste 4 Th.



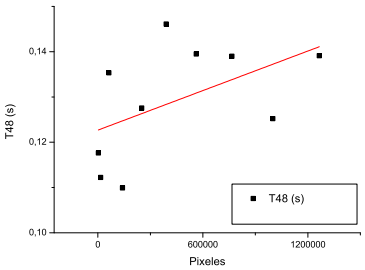
(d) Ajuste 8 Th.



(e) Ajuste 12 Th.



(f) Ajuste 24 Th.



(g) Ajuste 48 Th.

Figura 6.7: Ajustes para los diferentes tiempos respecto a la resolución variando el número de hilos de ejecución.

Puede verse que la relación tiempo/resolución sigue una ecuación líneal cuya pendiente depende del número de hilos y crece con la resolución. Este resultado no es todavía concluyente ya que aunque se está midiendo el tiempo de cálculo de la distancia Hausdroff para los contornos que se encuentren en las imágenes, por lo que esta comparación con la resolución total nos ayuda a inferir un comportamiento, la verificación del resultado se realiza con la segunda batería de pruebas. Como se puede ver en el cuadro 6.5, al aumentar el número de hilos de 1 a 12 la pendiente se verá reducida en la misma proporción en que aumentan los hilos para cada caso. Al cambiar de 12 a 14 hilos la pendiente casi no varía y en 48 hilos el comportamiento de la recta parece caótico. También puede apreciarse que la ordenada en el origen es prácticamente cero en todos los casos.

La pauta presentada por los tiempos correspondientes a los 24 hilos son casi idénticos a los resultados obtenidos para 12, este hecho se puede explicar porque OpenMP ha repartido las tareas solamente entre dos de los cuatro *sockets* disponibles para obtener los 24 hilos de ejecución, usando por tanto 12 hilos reales y 12 hilos virtuales, proporcionados por la técnica de HyperThreading de los procesadores Intel que ejecutan la prueba, ya que cada *socket* cuenta con una única CPU de seis núcleos con HT. En el caso de los tiempos medidos para 48 hilos, el comportamiento puede explicarse debido a la falta de tareas para ejecutar, dado que los contornos en esta prueba contienen pocos píxeles y por tanto se pierde más tiempo repartiendo las tareas entre los 48 hilos que el tiempo que se emplea en ejecutarlas.

Representación de los tiempos frente al número de píxeles del contorno para el segundo grupo de pruebas:

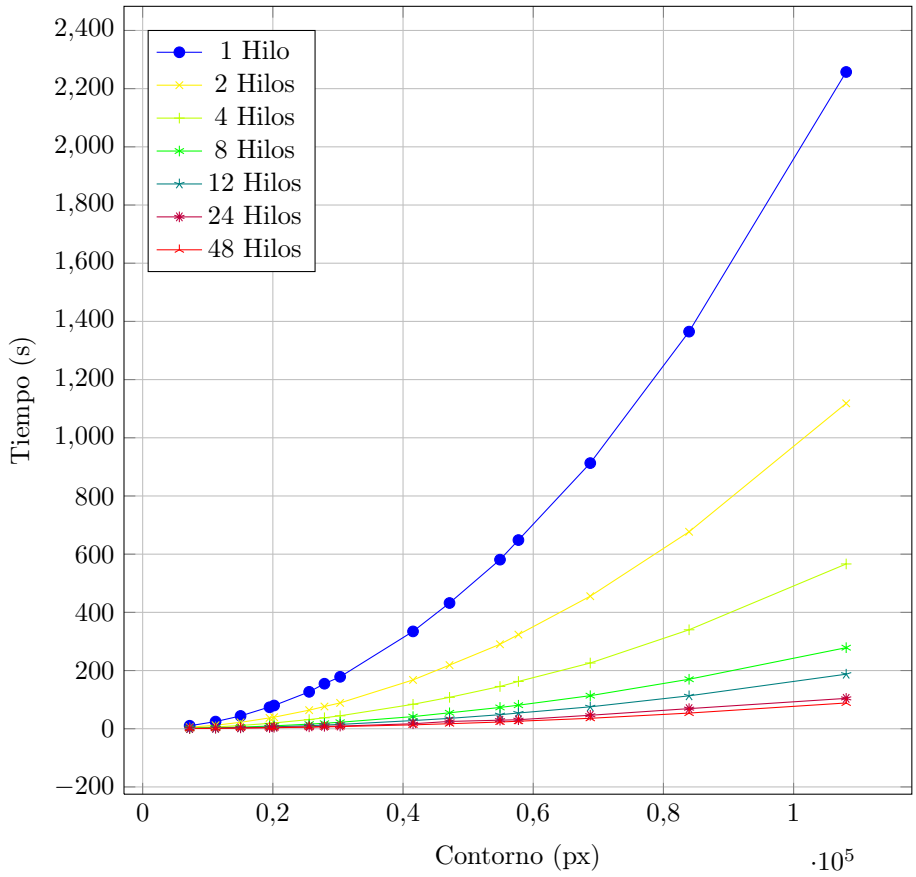
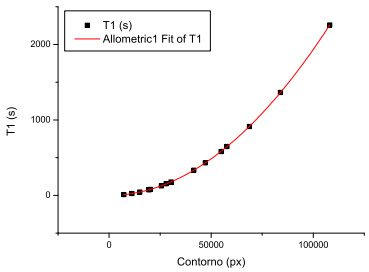


Figura 6.8: Tiempo frente a píxeles de contorno.

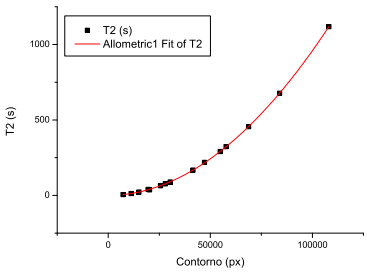
Resultados de los ajustes a la ecuación $y = a \cdot x^b$:

Hilos	a	b	r^2
1	$2,03778 \cdot 10^{-7} \pm 4,7332 \cdot 10^{-9}$	$1,99543 \pm 0,00204$	0,99999
2	$1,16422 \cdot 10^{-7} \pm 4,69871 \cdot 10^{-9}$	$1,98312 \pm 0,00354$	0,99998
4	$4,59282 \cdot 10^{-8} \pm 2,85375 \cdot 10^{-9}$	$2,00457 \pm 0,00545$	0,99995
8	$3,12671 \cdot 10^{-8} \pm 1,68337 \cdot 10^{-9}$	$1,97676 \pm 0,00472$	0,99996
12	$1,66313 \cdot 10^{-8} \pm 8,8261 \cdot 10^{-10}$	$1,99687 \pm 0,00465$	0,99996
24	$5,36606 \cdot 10^{-8} \pm 1,95403 \cdot 10^{-8}$	$1,84637 \pm 0,03201$	0,99784
48	$1,12987 \cdot 10^{-8} \pm 1,16434 \cdot 10^{-9}$	$1,96544 \pm 0,00904$	0,99985

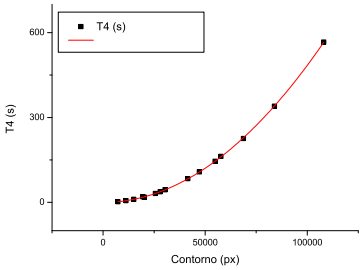
Tabla 6.6: Ajuste alométrico de los tiempos de ejecución respecto a la resolución para cada grupo de Hilos.



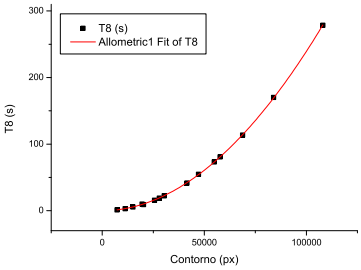
(a) Ajuste 1 Th.



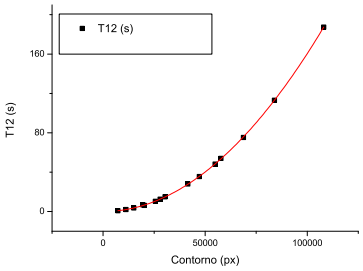
(b) Ajuste 2 Th.



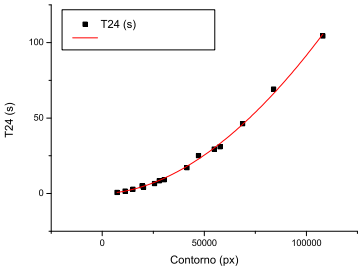
(c) Ajuste 4 Th.



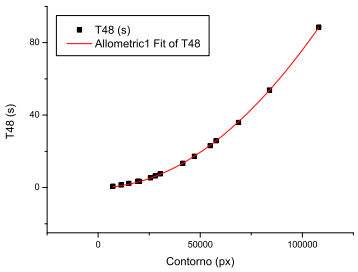
(d) Ajuste 8 Th.



(e) Ajuste 12 Th.



(f) Ajuste 24 Th.



(g) Ajuste 48 Th.

Figura 6.9: Ajustes para los diferentes tiempos respecto al número de píxeles de los contornos variando el número de hilos de ejecución.

Los valores obtenidos en este estudio muestran que el crecimiento del tiempo de procesamiento aumenta de forma exponencial con la variación del número de píxeles del contorno. Este crecimiento es aproximadamente cuadrático, ya que como se puede ver en la tabla 6.6 los valores de b son muy cercanos a 2. El comportamiento frente al aumento del número de hilos de ejecución es más dificil de deducir con estos resultados que en el caso anterior, ya que es el factor a el que varia, pero no parece que su comportamiento tenga relación con el número de hilos. Para poder establecer una patrón de comportamiento se representan a continuación para cada imagen los tiempos de proceso frente al número de hilos.

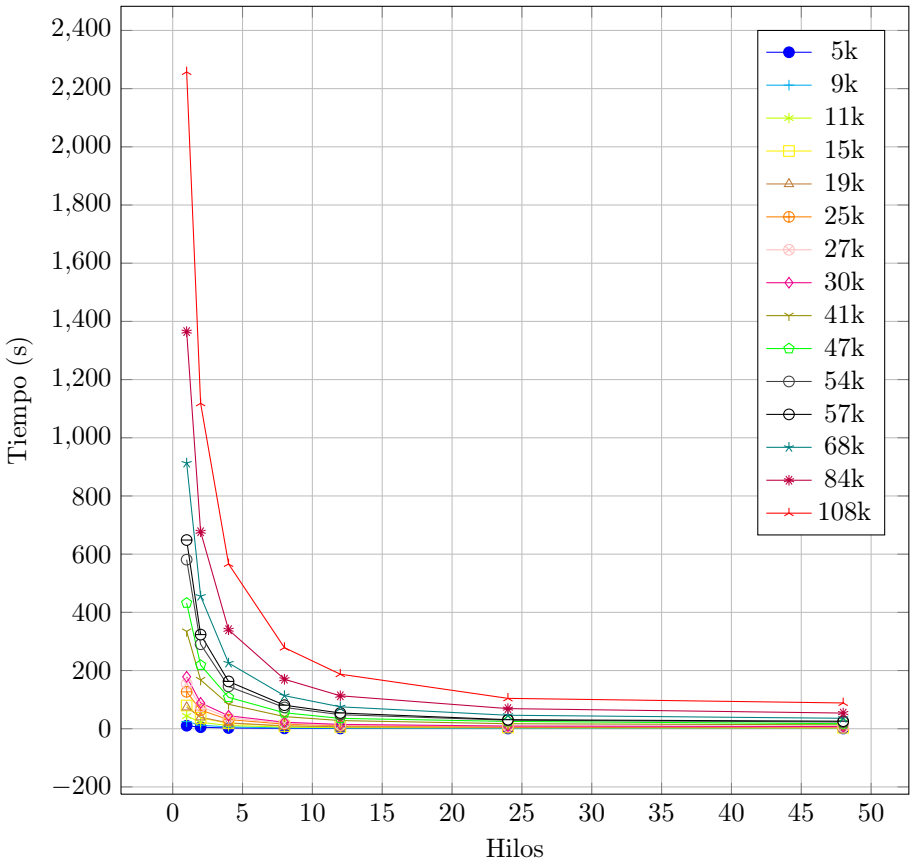
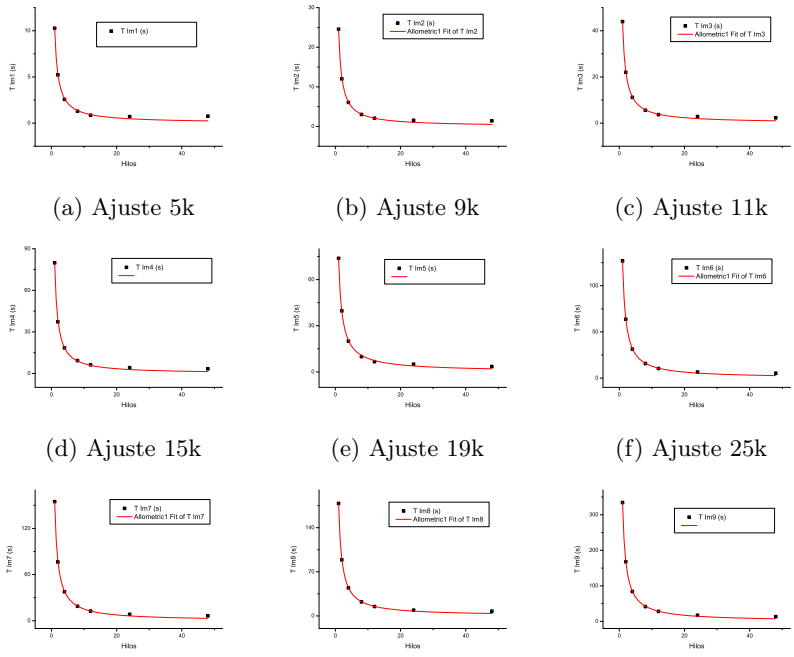


Figura 6.10: Tiempo frente a Hilos.

Los ajustes correspondientes a la figura 6.10 se muestran en la tabla 6.12 y en las figuras 6.12:

Imagen	a	b	r^2
5k	$10,24311 \pm 0,25208$	$-0,9666 \pm 0,04449$	0,99458
9k	$24,42901 \pm 0,46126$	$-0,99263 \pm 0,03533$	0,99686
11k	$43,8289 \pm 0,70278$	$-0,97549 \pm 0,02933$	0,99773
15k	$79,3642 \pm 1,21224$	$-1,04606 \pm 0,03062$	0,99799
19k	$73,92704 \pm 1,0176$	$-0,92445 \pm 0,02351$	0,99831
25k	$126,70268 \pm 1,27557$	$-0,99044 \pm 0,01878$	0,99912
27k	$154,36974 \pm 1,70587$	$-1,00386 \pm 0,02098$	0,99894
30k	$178,08939 \pm 1,784$	$-0,99247 \pm 0,01874$	0,99913
41k	$334,07634 \pm 2,98173$	$-0,98882 \pm 0,01662$	0,99931
47k	$431,7543 \pm 4,50089$	$-0,98321 \pm 0,01926$	0,99905
54k	$580,18812 \pm 5,12272$	$-0,99125 \pm 0,01649$	0,99932
57k	$647,49737 \pm 5,51414$	$-0,99291 \pm 0,01594$	0,99937
68k	$911,72725 \pm 8,16182$	$-0,99579 \pm 0,01682$	0,99931
84k	$1361,99025 \pm 12,33753$	$-0,99589 \pm 0,01702$	0,99929
108k	$2253,05501 \pm 19,01194$	$-0,99741 \pm 0,01589$	0,99939

Tabla 6.7: Ajuste alométrico de los tiempos de ejecución respecto a la resolución para cada grupo de Hilos.

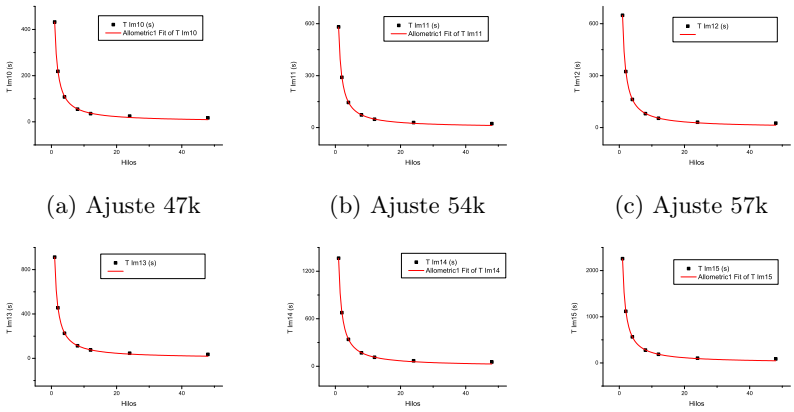


(a) Ajuste 5k (b) Ajuste 9k (c) Ajuste 11k

(d) Ajuste 15k (e) Ajuste 19k (f) Ajuste 25k

(g) Ajuste 27k (h) Ajuste 30k (i) Ajuste 41k

Figura 6.11: Ajustes correspondientes a la relacción entre el tiempo y el número de hilos de ejecución para cada imagen.



(a) Ajuste 47k (b) Ajuste 54k (c) Ajuste 57k

(d) Ajuste 68k (e) Ajuste 84k (f) Ajuste 108k

Figura 6.12: Ajustes correspondientes a la relacción entre el tiempo y el número de hilos de ejecución para cada imagen.

De los resultados del ajuste es posible deducir que, efectivamente, existe un patrón de comportamiento que rige el tiempo de proceso cuando se aumenta el número de hilos. Si comparamos los resultados de la tabla 6.12 para un solo hilo con los valores del parametro a de los ajustes, se ve que coinciden plenamente. Siempre que haya tareas suficiente para que la paralelización merezca la pena y no sucedan comportamientos como el mostrado en la gráfica 6.6, el tiempo de ejecución seguirá la siguiente fórmula:

$$t_{th} = t_1 \cdot th^{-1} \tag{6.3.0.2}$$

Donde t_{th} es el tiempo final de proceso, t_1 es el tiempo con un solo hilo y th el número de hilos donde deseamos conocer el comportamiento. Esta ecuación demuestra que el tiempo de ejecución se dividirá de forma proporcional al número de hilos reales que haya disponibles para realizar el cálculo de la distancia de Hausdorff. Es necesario hacer la matización de los hilos reales de ejecución porque el resultado con 48 hilos siendo 24 reales más 24 virtuales no sigue el patrón, aunque se obtenga también una mejora del rendimiento. Este resultado corrobora totalmente las predicciones teóricas presentadas en Amdahl (1967); Shameem Akhter (2006).

Capítulo 7

Conclusiones

Ésta última sección presenta las conclusiones extraídas del trabajo realizado y propone líneas de continuación para proyectos futuros.

7.1. Conclusiones

Como se ha visto en los capítulos anteriores, el trabajo ha cumplido satisfactoriamente los dos objetivos principales de la propuesta inicial:

- Se ha creado e implementado un algoritmo que sirve para el reconocimiento de patrones en imágenes haciendo uso de la Distancia de Hausdorff.
- Se ha demostrado la alta capacidad del algoritmo basado en la distancia de Hausdorff para ejecutarse de forma paralela, obteniendo mejoras notables en el rendimiento. En la tabla 7.1 se muestran la relación de aceleración de cada ejecución para el segundo grupo de pruebas con respecto al código secuencial:

.

Img	Tiempo (s)													
	1	speedup	2	speedup	4	speedup	8	speedup	12	speedup	24	speedup	48	speedup
5k	10,27	1	5,22	1,97	2,57	3,99	1,30	7,93	0,86	11,88	0,70	14,60	0,76	13,53
9k	24,55	1	12,02	2,04	6,08	4,04	3,03	8,11	2,07	11,85	1,55	15,79	1,41	17,45
11k	43,97	1	22,04	2,00	11,16	3,94	5,57	7,89	3,71	11,86	2,86	15,36	2,30	19,12
15k	79,81	1	39,68	2,01	20,04	3,98	9,96	8,01	6,72	11,88	5,08	15,72	3,56	22,43
19k	73,77	1	37,25	1,98	18,52	3,98	9,29	7,94	6,27	11,77	4,22	17,48	3,41	21,61
25k	126,80	1	63,85	1,99	31,46	4,03	15,82	8,01	10,42	12,17	6,56	19,34	5,34	23,73
27k	154,71	1	76,41	2,02	37,72	4,10	18,87	8,20	12,61	12,27	8,41	18,39	6,38	24,25
30k	178,45	1	88,84	2,01	44,54	4,01	22,27	8,01	14,94	11,95	9,15	19,50	7,52	23,73
41k	334,51	1	167,75	1,99	83,99	3,98	41,70	8,02	28,07	11,92	17,16	19,49	13,39	24,98
47k	432,15	1	218,66	1,98	107,90	4,00	54,80	7,89	35,64	12,13	25,14	17,19	17,29	25,00
54k	581,11	1	290,32	2,00	145,30	4,00	73,16	7,94	48,24	12,05	29,38	19,78	23,12	25,13
57k	648,47	1	323,51	2,00	162,58	3,99	80,91	8,02	54,06	12,00	31,02	20,91	25,85	25,08
68k	913,02	1	455,55	2,00	225,88	4,04	113,65	8,03	75,29	12,13	46,31	19,71	35,91	25,43
84k	1364,87	1	676,97	2,02	340,14	4,01	170,13	8,02	113,14	12,06	69,15	19,74	53,82	25,36
108k	2257,29	1	1118,41	2,02	566,37	3,99	278,62	8,10	187,43	12,04	104,36	21,63	88,57	25,48
$\overline{X}$		**1**		**2,00**		**4.00**		**8,00**		**11,99**		**18.30**		**22.82**

Tabla 7.1: Relación entre los diferentes tiempos de ejecución.

Y en la figura 7.1la representación de la aceleración media entre todas las imágenes frente al número de hilos:

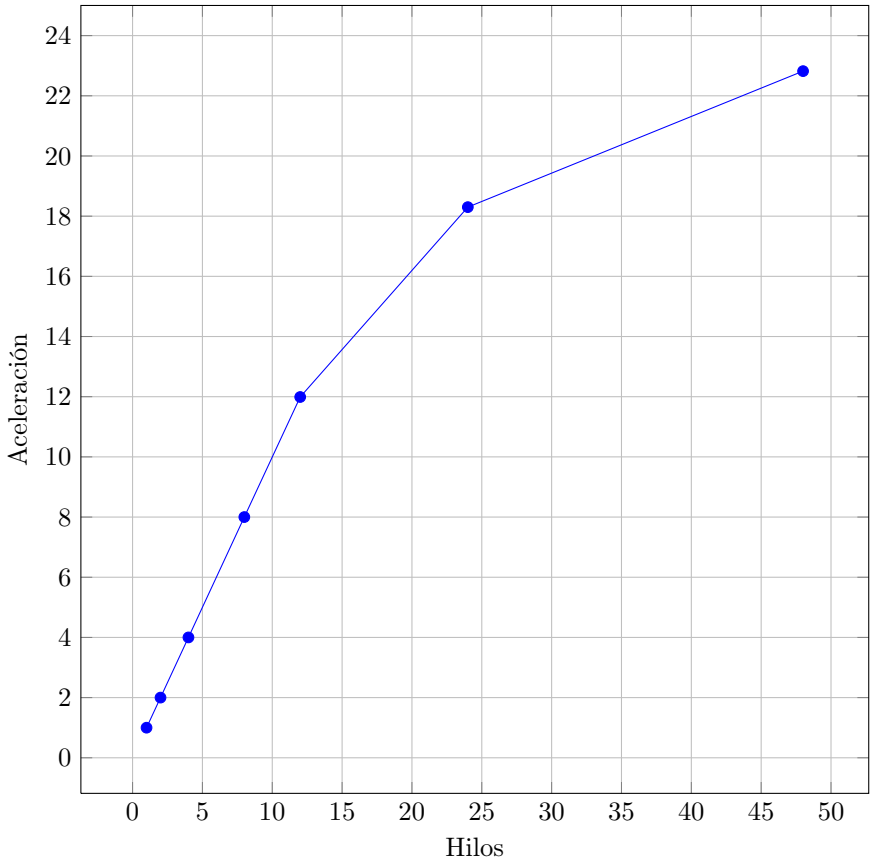


Figura 7.1: Aceleración frente a Hilos.

Puede verse que la aceleración de la ejecución crece de forma lineal hasta los 12 núcleos, en 24 se frena pero sigue habiendo una mejora notable, y el caso de 48 ya está explicado por tratarse de 24 hilos reales más 24 hilos virtuales. Como se puede ver en la tabla 7.1, para las imágenes con menor número de píxeles la aceleración es menor, pero ésta aumenta a medida que hay más tareas que repartir, por tanto también queda patente que la relación entre la cantidad de datos que se quieren tratar y la capacidad de computación que se aplique sobre estos deben mantener una cierta proporción. Otra conclusión directa que puede extraerse de este resultado es que, según la Ley de Amdahl, el porcentaje de la carga de trabajo del código secuencial es mínima en proporción con la carga de trabajo de la sección paralela, de otra forma no sería posible obtener unas aceleraciones tan cercanas al modelo teórico de Amdahl Amdahl (1967).

Éste último resultado es especialmente importante, ya que demuestra que las predicciones teóricas de ganancia en rendimiento, cuando la paralelización de tareas es posible, pueden alcanzarse en la práctica.

Estudiando los resultados, se puede inferir que a la hora de seleccionar imágenes y máquinas sobre las que analizarlas los dos factores a tener en cuenta

son:

- El tiempo de proceso varía de forma cuadrática cuando aumenta el número de píxeles de los contornos.
- El tiempo de ejecución varía inversamente con respecto al número de hilos reales donde se ejecute.

Para el análisis con la distancia de Hausdorff, se ha obtenido que el tiempo de proceso aumenta de forma cuadrática al aumentar el número de elementos de los grupos a comparar, y hemos visto que este algoritmo paralelizado tiene una mejora de tiempo proporcional al número de hilos de ejecución disponibles. La figura 7.2 presenta una comparación entre t_{th}^{-1} y t_1^2 y su producto:

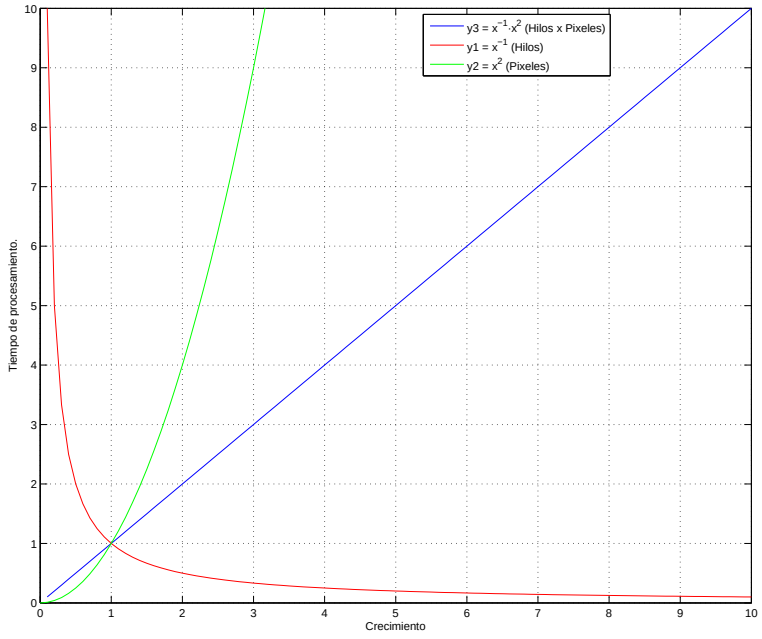


Figura 7.2: t_{th}^{-1}, t_1^2 y $t_{th}^{-1} \cdot t_1^2$

Tenemos entonces, que la modificación del tiempo de proceso al cambiar el número de píxeles de los contornos a comparar variará de forma lineal siempre que se ajusten los hilos de ejecución en la misma proporción. Éste es un resultado esperado, ya que obtener una disminución del tiempo de proceso que siga la teoría, t_{th}^{-1} al aumentar la capacidad de procesamiento paralelo, es extremadamente difícil Amdahl (1967); Shameem Akhter (2006) y en el caso del algoritmo presentado, se ha demostrado que el aumento del tiempo de computación se comporta como t^2 al variar el número de píxeles de los contornos. Por tanto si se varían el número de hilos en la misma proporción en que cambie el número de píxeles $t_{th}^{-1} \cdot {t_1}^2 = t_{th}^1$ el resultado será una variación lineal del tiempo de proceso.

La obtención de una mejora lineal con el aumento de hilos de ejecución se puede explicar para este algoritmo, porque aunque se tienen que repartir muchas tareas que requieren instrucciones sencillas, puesto que solamente se realizan operaciones aritméticas básicas, y no es necesario operar con vectores o matrices en ningún caso. Esto hace que la división sea posible a partir de imágenes muy pequeñas y que las diferencias de reparto de tareas no tengan un gran peso, al ser cada tarea de muy rápida ejecución, de forma que los tiempos de espera serán mínimos. Puede no obstante ocurrir que, según el experimento, el resultado ofrezca un rendimiento temporal peor; vease los datos de la tabla 6.10 para el caso de 48 hilos, debido a que el tiempo dedicado a repartir las tareas entre el número de hilos empleados para resolver el problema planteado supere al tiempo de ejecución repartido entre los diferentes hilos. Se hace primordial entonces, optimizar el número de hilos necesarios para ejecutar la tarea de forma que el tiempo de proceso sea el menor posible y no se produzcan esos retrasos por la distribución y organización de las tareas.

También se concluye que la elección del uso de OpenMP para realizar la paralelización de la versión secuencial del algoritmo ha sido acertada y que para este caso la planificación estática genera unos resultados difícilmente mejorables. Esto demuestra la eficacia del compilador de OpenMP para repartir las tareas de forma equitativa. A la vista de este comportamiento de OpenMP puede deducirse también que el algoritmo presentado puede no ser exclusivamente paralelizado según tareas o threads, como en este trabajo, sino siguiendo el paradigma SIMD, ya que se ejecutan una gran cantidad de tareas sencillas sobre datos diferentes, teniendo siempre en cuenta que la ganancia de rendimiento depende directamente de la reacción entre el nivel de paralelización y la cantidad de datos con los que trabajar.

7.2. Trabajos futuros

Se proponen la siguiente lista de líneas de trabajo para continuar el estudio aquí iniciado:

- Realizar una implementación SIMD usando CUDA para observar el comportamiento con un nivel mayor de paralelización en imágenes de más resolución.
- Modificar el algoritmo para disminuir el número de puntos de los contornos, de forma que hagan falta menos elementos para el cálculo de la distancia.
- Estudiar si tomar sólo los vértices es una opción viable.
- Ampliar el programa incluyendo una función que permita descartar imágenes en base a la distancia obtenida.
- Cálculo automático del límite para el que se consideran suficientemente iguales los contornos comparados.
- Creación de un algoritmo que convolucione un contorno sobre una imagen mayor y permita encontrar coincidencias, permitiendo así el reconocimiento de patrones pequeños en imágenes grandes.

- Estudio de viabilidad de uso de la distancia Hausdorff para el reconocimiento de figuras en 3D y su posible aplicación para su uso en sistemas de visión artificial.

Apéndice A

Compilación y ejecución del programa

Es necesario tener las OpenCV instaladas y configuradas para que pueda compilarse y ejecutarse el programa. A continuación se muestra como obener una versión de las OpenCV en el sistema operativo Ubuntu Linux, para otras versiones consultar en http://opencv.willowgarage.com/wiki/.

Ejecutar en la consola los siguientes comandos:

- sudo apt-get install cmake
- sudo apt-get install g++
- wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.2/OpenCV-2.4.2.tar.bz2/download
- mv download OpenCV-2.4.2.tar.bz2
- tar xjvf OpenCV-2.4.2.tar.bz2
- cd OpenCV-2.4.2
- cmake -DCMAKE_BUILD_TYPE=RELEASE -DBUILD_PYTHON_SUPPORT=ON -D:OPENCV_BUILD_3RDPARTY_LIBS=ON -DUSE_SSE=ON -DUSE_SSE2=ON -DUSE_SSE3=ON -DCMAKE_INSTALL_PREFIX=$HOME/opt/opencv
- make -j<numero de cores>
- make install
- export LD_LIBRARY_PATH=/usr/local/lib

Esta secuencia deja las OpenCV instaladas en $HOME/opt/opencv y el *path* configurado apra que el compilador pueda encontrarlas.

Una vez hecho esto se puede compilar el programa ejecutando el *Makefile* incluido en la carpeta raíz del proyecto. Este *Makefile* tiene el siguiente aspecto:

```
OPENCV_HOME=$HOME/opt/opencv
#PKG_CONFIG_PATH=/home/lmsan/opt/opencv/lib/pkgconfig/:$PKG_CONFIG_PATH
INCLUDES=-I$OPENCV_HOME/include
LIBRARIES=-L$OPENCV_HOME/lib
#LIBS=-lm -lcxcore -lcv -lhighgui -lcvaux
LIBS=-lm -lopencv_core -lopencv_imgproc -lopencv_highgui
#PK='pkg-config –cflags opencv' 'pkg-config –libs opencv'

all: mkdir -p bin g++ src/aux_functions.cpp -c -o bin/aux_functions.o $(INCLUDES) $(LIBRARIES) $(LIBS) $(PK) g++ src/coords.cpp -c -o bin/coords.o $(INCLUDES) $(LIBRARIES) $(LIBS) $(PK) g++ src/hausdorff_single.cpp -c -o bin/hausdorff_single.o $(INCLUDES) g++ src/hausdorff_omp.cpp -fopenmp -c -o bin/hausdorff_omp.o $(INCLUDES) g++ src/main_single.cpp -c -o bin/main_single.o $(INCLUDES) $(LIBRARIES) $(LIBS) $(PK) g++ src/main_omp.cpp -c -o bin/main_omp.o $(INCLUDES) $(LIBRARIES) $(LIBS) $(PK) g++ bin/aux_functions.o bin/coords.o bin/hausdorff_single.o bin/main_single.o -o main_single $(INCLUDES) $(LIBRARIES) $(LIBS) $(PK) g++ bin/aux_functions.o bin/coords.o bin/hausdorff_omp.o bin/main_omp.o -fopenmp -o main_omp $(INCLUDES) $(LIBRARIES) $(LIBS) $(PK) clean: rm -rf bin *.o rm main*
```

La ejecución se hará llamando al siguiente comando:

- ./main_<version> -t ../../img/testX/<Im1>.jpg -m ../../img/testX/<Im2>.jpg

Donde <version> es la versión del programa para ejecutar: Single o OMP, textX la carpeta de imágenes correspondiente al test,se el A o el B y <Im1> y <Im2> el nombre de las imágenes. -t y -m son los *flags* del programa, estos son obligatorios para que este reconozca que imagen es el *template* y cual el *map* sobre el que hacer el reconocimiento.

Apéndice B

Resultados experimentales

B.1. *Scripts* de experimentación

Los *scripts* de experimentación han seguido el siguiente formato:

```
for (i = 0 i < 10; i++)
do
   ../.././ main\_<version> -t ../../img/testX/<Im1>.jpg
    -m ../../img/testX/<Im2>.jpg
done
```

Listing B.1: border_detection()

Donde <version> es la versión del programa para ejecutar: Single o OMP, textX la carpeta de imágenes correspondiente al test,se el A o el B y <Im1> y <Im2> el nombre de las imágenes. Para la versión de OpenMP, hay que limitar el número de hilos disponibles, esto se hace ejecutando el comando export OMP_NUM_THREADS=<número de threads> en la consola antes de llamar al programa.

Se han usado ademas cuatro scripts que ejecutan todos los scripts necesarios para realizar todas las pruebas del grupo de tests A y B en paralelo y con un solo hilo. Por ejemplo:

```
echo "--- 50 ---"
sh A50s.sh
echo "--- 125 ---"
sh A125s.sh
echo "--- 250 ---"
sh A250s.sh
echo "--- 375 ---"
sh A375s.sh
echo "--- 500 ---"
sh A500s.sh
echo "--- 625 ---"
sh A625s.sh
echo "--- 750 ---"
sh A750s.sh
echo "--- 875 ---"
sh A875s.sh
echo "--- 1000 ---"
sh A1000s.sh
echo "--- 1125 ---"
sh A1125s.sh
```

Listing B.2: border_detection()

El volcado de datos se realizará, en una consola de tipo *bash* llamando al script de la siguiente forma: sh script.sh > resultados.txt.

B.2. Test A

B.2.1. 1 Hilo

— 50 — T: 0,00156999 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00157189 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00220394 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00157285 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00163007 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00157404 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00163317 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00157309 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00157189 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00157213 D: 5,83095 PEIm1: 86 PEIm2: 89 — 125 — T: 0,0084331 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00843811 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00841093 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00841999 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,008425 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00842118 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00841999 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00841999 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,008425 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00853109 D: 18,868 PEIm1: 212 PEIm2: 198 — 250 — T: 0,0315239 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0302579 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0302351 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0302541 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0302448 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,030014 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,02986 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,030159 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0302501 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,030221 D: 37,4833 PEIm1: 391 PEIm2: 387 — 375 — T: 0,0659199 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,062567 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,062541 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0658872 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,065619 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0664909 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,062547 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,062644 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,062768 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0625319 D: 55,9017 PEIm1: 574 PEIm2: 554 — 500 — T: 0,118783 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,112671 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,120928 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,116962 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,119836 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,118704 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,112705 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,11781 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,112688 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,112764 D: 75,1665 PEIm1: 763 PEIm2: 751 — 625 — T: 0,173691 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,187297 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,174713 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,174453 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,175106 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,171775 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,174438 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,174189 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,174434 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,176625 D: 91,2195 PEIm1: 951 PEIm2: 934 — 750 — T: 0,254875 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,246699 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,246749 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,268328 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,2566 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,246855 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,254897 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,247549 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,246838 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,242984 D: 111,946 PEIm1: 1126 PEIm2: 1112 — 875 — T: 0,347357 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,34447 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,329108 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,332402 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,353144 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,332484 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,332381 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,332283 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,356508 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,332255 D: 132,246 PEIm1: 1310 PEIm2: 1293 — 1000 — T: 0,435502 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,436235 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,435831 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,435878 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,432751 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,436051 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,453815 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,440309 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,433063 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,435943 D: 150,655 PEIm1: 1497 PEIm2: 1480 — 1125 — T: 0,54394 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,544908 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,544224 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,544237 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,584359 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,544209 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,541206 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,54408 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,550805 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,551447 D: 167,26 PEIm1: 1673 PEIm2: 1652

B.2.2. 2 Hilos

— 50 — T: 0,00102711 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,0010159 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,0010798 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,0010469 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00103188 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00102186 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00106215 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00103211 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00102496 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00101304 D: 5,83095 PEIm1: 86 PEIm2: 89 — 125 — T: 0,00450182 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00449395 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00471401 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00450802 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,0048039 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00688004 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00453305 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00450706 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00458002 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00455594 D: 18,868 PEIm1: 212 PEIm2: 198 — 250 — T: 0,0155301 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0156782 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,015511 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,015697 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0156879 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0163829 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,015856 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0163109 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,015518 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,015507 D: 37,4833 PEIm1: 391 PEIm2: 387 — 375 — T: 0,0321331 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,031611 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,031625 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,031589 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0315979 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,031584 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,031563 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0315981 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0316749 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,031569 D: 55,9017 PEIm1: 574 PEIm2: 554 — 500 — T: 0,0566831 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0598302 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,056684 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,056638 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0566559 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,056673 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0576341 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0566878 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,05672 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,05667 D: 75,1665 PEIm1: 763 PEIm2: 751 — 625 — T: 0,088985 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,087594 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0876789 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0876908 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,092545 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0927298 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,08762 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0876868 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,087697 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,087677 D: 91,2195 PEIm1: 951 PEIm2: 934 — 750 — T: 0,124157 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,124139 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,126129 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,127023 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,130922 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,126774 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,123822 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,124152 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,124074 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,127051 D: 111,946 PEIm1: 1126 PEIm2: 1112 — 875 — T: 0,166581 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,166681 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,167402 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,166746 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,166767 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,16652 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,16682 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,166867 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,166575 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,17214 D: 132,246 PEIm1: 1310 PEIm2: 1293 — 1000 — T: 0,218675 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,21934 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,221971 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,21909 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,219342 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,219345 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,21921 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,219212 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,219329 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,219236 D: 150,655 PEIm1: 1497 PEIm2: 1480 — 1125 — T: 0,288255 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,276293 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,274234 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,275145 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,273962 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,274169 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,274103 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,289383 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,274319 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,274436 D: 167,26 PEIm1: 1673 PEIm2: 1652

B.2.3. 4 Hilos

— 50 — T: 0,000724077 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000725985 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000749111 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000733137 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000730038 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000756979 D: 5,83095 PEIm1:

86 PEIm2: 89 T: 0,000739098 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000678062 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000743866 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000729084 D: 5,83095 PEIm1: 86 PEIm2: 89 — 125 — T: 0,00244212 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00243211 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00246191 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00247717 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00245094 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00241303 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00246096 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00247383 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00242114 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00242591 D: 18,868 PEIm1: 212 PEIm2: 198 — 250 — T: 0,00800395 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00801516 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00798297 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00797391 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,007967 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00795388 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00795603 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00801682 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00798607 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00798106 D: 37,4833 PEIm1: 391 PEIm2: 387 — 375 — T: 0,0159998 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0162692 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,016274 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0159719 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0159762 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0159888 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0160031 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,016897 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0163798 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0160239 D: 55,9017 PEIm1: 574 PEIm2: 554 — 500 — T: 0,029547 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,028616 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0301449 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0293031 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0301921 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0291071 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,028615 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0286009 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0286129 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,02876 D: 75,1665 PEIm1: 763 PEIm2: 751 — 625 — T: 0,0440838 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0441551 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,044924 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0449438 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0441251 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,04409 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,044101 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,045049 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0452089 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,044323 D: 91,2195 PEIm1: 951 PEIm2: 934 — 750 — T: 0,063412 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,062324 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,062748 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0624621 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0623469 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,062464 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,062459 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0656772 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,062295 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0624049 D: 111,946 PEIm1: 1126 PEIm2: 1112 — 875 — T: 0,0839331 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0863779 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0839739 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0839341 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,085412 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0839939 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,083957 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0840161 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,083936 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0839598 D: 132,246 PEIm1: 1310 PEIm2: 1293 — 1000 — T: 0,110481 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,111243 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,112173 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,110418 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,110503 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,116242 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,110388 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,11047 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,110214 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,110467 D: 150,655 PEIm1: 1497 PEIm2: 1480 — 1125 — T: 0,137588 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,137828 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,137615 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,137862 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,14538 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,142625 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,1378 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,138347 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,1378 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,139213 D: 167,26 PEIm1: 1673 PEIm2: 1652

B.2.4. 8 Hilos

— 50 — T: 0,000597 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,0006001 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000633001 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000603914 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000578165 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000582933 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000619173 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000571012 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000585079 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000566006 D: 5,83095 PEIm1: 86 PEIm2: 89 — 125 — T: 0,00160003 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00147986 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00149322 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,0014379 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00151396 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00148892 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00149298 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00145197 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00143003 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00146103 D: 18,868 PEIm1: 212 PEIm2: 198 — 250 — T: 0,00459504 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00440907 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00429797 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,004637 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00590801 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0043571 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0046699 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0042491 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00441408 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00426888 D: 37,4833 PEIm1: 391 PEIm2: 387 — 375 — T: 0,0083611 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00868297 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00827885 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00833297 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00827503 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00852013 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00830197 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00833607 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,0083189 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00839806 D: 55,9017 PEIm1: 574 PEIm2: 554 — 500 — T: 0,014668 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0146642 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0146132 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0147951 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0241199 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0148089 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,014622 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0146191 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0145688 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0146499 D: 75,1665 PEIm1: 763 PEIm2: 751 — 625 — T: 0,022712 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0223908 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0226948 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0224152 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0238891 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0235691 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,023607 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0233819 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,023005 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0235782 D: 91,2195 PEIm1: 951 PEIm2: 934 — 750 — T: 0,0332808 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0314128 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,031424 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0314269 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0315549 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0314479 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0314078 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,031517 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0313911 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,03318 D: 111,946 PEIm1: 1126 PEIm2: 1112 — 875 — T: 0,0424759 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0429471 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,042552 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,042506 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,043277 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,042346 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0424881 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0423651 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,042573 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0423431 D: 132,246 PEIm1: 1310 PEIm2: 1293 — 1000 — T: 0,055845 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0589089 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0557961 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0557001 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0558369 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0557899 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0569279 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0557621 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0557861 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0557492 D: 150,655 PEIm1: 1497 PEIm2: 1480 — 1125 — T: 0,0693729 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0705669 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,069474 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,069411 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0699148 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,069392 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0693591 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0710661 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,069824 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,070533 D: 167,26 PEIm1: 1673 PEIm2: 1652

B.2.5. 12 Hilos

— 50 — T: 0,000741959 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000740051 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000696898 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000731945 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000697851 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000756025 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000674009 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00072813 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000658035 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,000673056 D: 5,83095 PEIm1: 86 PEIm2: 89 — 125 — T: 0,00122404 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00121713 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00122881 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00132895 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00119901 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00131583 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00117993 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00137401 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00117612 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00143194 D: 18,868 PEIm1:

212 PEIm2: 198 — 250 — T: 0,00361109 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00323606 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,003371 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00335288 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00312591 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0031271 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00320506 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,0033989 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00336981 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00340986 D: 37,4833 PEIm1: 391 PEIm2: 387 — 375 — T: 0,00577784 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00580096 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00577283 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00581717 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00577092 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00605893 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00580812 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00578499 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00576401 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00593305 D: 55,9017 PEIm1: 574 PEIm2: 554 — 500 — T: 0,0100629 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,010097 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0100479 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0105619 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,010093 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,010186 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,010026 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0100901 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0102971 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,01003 D: 75,1665 PEIm1: 763 PEIm2: 751 — 625 — T: 0,0153689 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,01564 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0155952 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0155439 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0156331 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,015511 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0152471 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0151961 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0152559 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0155129 D: 91,2195 PEIm1: 951 PEIm2: 934 — 750 — T: 0,02127 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0228629 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0213878 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0213699 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0216179 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0212979 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0213809 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0214281 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,021307 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0213501 D: 111,946 PEIm1: 1126 PEIm2: 1112 — 875 — T: 0,0291729 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0290051 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0291479 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0287402 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,028717 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0286911 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,028718 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0290771 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0287271 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0286999 D: 132,246 PEIm1: 1310 PEIm2: 1293 — 1000 — T: 0,0404129 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,037539 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,037564 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0384049 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,037586 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,03808 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0375822 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0375102 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0375161 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,037524 D: 150,655 PEIm1: 1497 PEIm2: 1480 — 1125 — T: 0,0469179 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0469029 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,046978 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,046859 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,047071 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0468912 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,046804 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,046891 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0469382 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0477669 D: 167,26 PEIm1: 1673 PEIm2: 1652

B.2.6. 24 Hilos

— 50 — T: 0,00208187 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00178385 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00224209 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00186205 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00160003 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00230598 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,0017221 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00150704 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00190687 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,00157094 D: 5,83095 PEIm1: 86 PEIm2: 89 — 125 — T: 0,00235105 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00187993 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00201297 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00212097 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00200319 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,002105 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00193191 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00178385 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,0022831 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,00151801 D: 18,868 PEIm1: 212 PEIm2: 198 — 250 — T: 0,00274086 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00416303 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00376487 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00288582 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00383306 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00377798 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00384593 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00400496 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,003896 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,00377107 D: 37,4833 PEIm1: 391 PEIm2: 387 — 375 — T: 0,00687504 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00626922 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00651193 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00658989 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00625181 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00499201 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00632811 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00508499 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00687885 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,00655198 D: 55,9017 PEIm1: 574 PEIm2: 554 — 500 — T: 0,00893688 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,010196 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,00893188 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,010592 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0102029 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0103819 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,010746 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,0101759 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,00928903 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,00933695 D: 75,1665 PEIm1: 763 PEIm2: 751 — 625 — T: 0,014888 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0151958 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0142372 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,014832 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,015559 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0156739 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,0140381 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,015296 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,015095 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,014607 D: 91,2195 PEIm1: 951 PEIm2: 934 — 750 — T: 0,0208249 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,020498 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,01882 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0200338 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0204279 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0215161 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,019428 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0190501 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,021687 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,02073 D: 111,946 PEIm1: 1126 PEIm2: 1112 — 875 — T: 0,027281 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0256748 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,028486 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,026933 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0270941 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0283439 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0270162 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,0272121 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,025696 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,027354 D: 132,246 PEIm1: 1310 PEIm2: 1293 — 1000 — T: 0,0328979 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,036876 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,03496 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0351858 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,034826 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0347512 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0349691 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,035074 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,034961 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,0348279 D: 150,655 PEIm1: 1497 PEIm2: 1480 — 1125 — T: 0,0663011 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0431042 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0455818 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0433121 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0396638 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,042846 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,043118 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0420799 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0448091 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0426769 D: 167,26 PEIm1: 1673 PEIm2: 1652

B.2.7. 48 Hilos

— 50 — T: 0,129075 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,109793 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,128606 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,050894 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,139563 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,129124 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,130971 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,140601 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,118911 D: 5,83095 PEIm1: 86 PEIm2: 89 T: 0,098752 D: 5,83095 PEIm1: 86 PEIm2: 89 — 125 — T: 0,125779 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,128735 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,137498 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,0373371 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,09707 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,078196 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,119612 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,0782771 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,18218 D: 18,868 PEIm1: 212 PEIm2: 198 T: 0,13759 D: 18,868 PEIm1: 212 PEIm2: 198 — 250 — T: 0,120626 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,060786 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,128636 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,160346 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,159132 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,300606 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,054111 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,138713 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,111477 D: 37,4833 PEIm1: 391 PEIm2: 387 T: 0,118963 D: 37,4833 PEIm1: 391 PEIm2: 387 — 375 — T: 0,0189979 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,137082 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,126668 D: 55,9017 PEIm1: 574 PEIm2: 554 T:

0,0682421 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,121147 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,129092 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,107594 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,136623 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,116197 D: 55,9017 PEIm1: 574 PEIm2: 554 T: 0,137539 D: 55,9017 PEIm1: 574 PEIm2: 554 — 500 — T: 0,138545 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,11993 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,139347 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,130102 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,128629 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,130816 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,110854 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,127351 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,141151 D: 75,1665 PEIm1: 763 PEIm2: 751 T: 0,108856 D: 75,1665 PEIm1: 763 PEIm2: 751 — 625 — T: 0,164605 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,130104 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,155685 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,14391 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,147452 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,124651 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,124393 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,129226 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,155205 D: 91,2195 PEIm1: 951 PEIm2: 934 T: 0,185717 D: 91,2195 PEIm1: 951 PEIm2: 934 — 750 — T: 0,108311 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,0426049 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,293981 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,127951 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,141967 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,138392 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,14192 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,137512 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,130742 D: 111,946 PEIm1: 1126 PEIm2: 1112 T: 0,131701 D: 111,946 PEIm1: 1126 PEIm2: 1112 — 875 — T: 0,129156 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,109679 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,141982 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,14702 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,158808 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,157332 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,160033 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,115386 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,139964 D: 132,246 PEIm1: 1310 PEIm2: 1293 T: 0,130565 D: 132,246 PEIm1: 1310 PEIm2: 1293 — 1000 — T: 0,122033 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,176144 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,111789 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,129877 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,129325 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,122801 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,121912 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,110465 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,123367 D: 150,655 PEIm1: 1497 PEIm2: 1480 T: 0,104382 D: 150,655 PEIm1: 1497 PEIm2: 1480 — 1125 — T: 0,162497 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,158732 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,150023 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0845778 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,130272 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,160825 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,148136 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,159402 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,0920889 D: 167,26 PEIm1: 1673 PEIm2: 1652 T: 0,144553 D: 167,26 PEIm1: 1673 PEIm2: 1652

B.3. Test B

B.3.1. 1 Hilo

— 5k — T: 10,1485 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,9613 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,286 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,1859 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,1862 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,158 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,2506 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,1862 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,1845 D: 0 PEIm1: 7219 PEIm2: 7219 T: 10,1901 D: 0 PEIm1: 7219 PEIm2: 7219 — 9k — T: 25,7525 D: 0 PEIm1: 11198 PEIm2: 11198 T: 23,8448 D: 0 PEIm1: 11198 PEIm2: 11198 T: 24,3959 D: 0 PEIm1: 11198 PEIm2: 11198 T: 24,2016 D: 0 PEIm1: 11198 PEIm2: 11198 T: 24,056 D: 0 PEIm1: 11198 PEIm2: 11198 T: 24,1172 D: 0 PEIm1: 11198 PEIm2: 11198 T: 25,1125 D: 0 PEIm1: 11198 PEIm2: 11198 T: 24,055 D: 0 PEIm1: 11198 PEIm2: 11198 T: 24,1428 D: 0 PEIm1: 11198 PEIm2: 11198 T: 25,8369 D: 0 PEIm1: 11198 PEIm2: 11198 — 11k — T: 43,6182 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,7658 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,6225 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,7456 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,5991 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,7962 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,6205 D: 0 PEIm1: 15040 PEIm2: 15040 T: 46,8513 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,7154 D: 0 PEIm1: 15040 PEIm2: 15040 T: 43,401 D: 0 PEIm1: 15040 PEIm2: 15040 — 15k — T: 78,3585 D: 0 PEIm1: 20189 PEIm2: 20189 T: 81,7964 D: 0 PEIm1: 20189 PEIm2: 20189 T: 83,9421 D: 0 PEIm1: 20189 PEIm2: 20189 T: 78,3694 D: 0 PEIm1: 20189 PEIm2: 20189 T: 79,0747 D: 0 PEIm1: 20189 PEIm2: 20189 T: 78,3605 D: 0 PEIm1: 20189 PEIm2: 20189 T: 78,1987 D: 0 PEIm1: 20189 PEIm2: 20189 T: 77,7211 D: 0 PEIm1: 20189 PEIm2: 20189 T: 78,3898 D: 0 PEIm1: 20189 PEIm2: 20189 T: 83,9317 D: 0 PEIm1: 20189 PEIm2: 20189 — 19k — T: 73,0435 D: 0 PEIm1: 19490 PEIm2: 19490 T: 73,3516 D: 0 PEIm1: 19490 PEIm2: 19490 T: 73,2284 D: 0 PEIm1: 19490 PEIm2: 19490 T: 75,4128 D: 0 PEIm1: 19490 PEIm2: 19490 T: 73,2351 D: 0 PEIm1: 19490 PEIm2: 19490 T: 76,4444 D: 0 PEIm1: 19490 PEIm2: 19490 T: 73,2748 D: 0 PEIm1: 19490 PEIm2: 19490 T: 73,2348 D: 0 PEIm1: 19490 PEIm2: 19490 T: 73,2368 D: 0 PEIm1: 19490 PEIm2: 19490 T: 73,2223 D: 0 PEIm1: 19490 PEIm2: 19490 — 25k — T: 125,05 D: 0 PEIm1: 25576 PEIm2: 25576 T: 125,083 D: 0 PEIm1: 25576 PEIm2: 25576 T: 130,85 D: 0 PEIm1: 25576 PEIm2: 25576 T: 125,075 D: 0 PEIm1: 25576 PEIm2: 25576 T: 125,239 D: 0 PEIm1: 25576 PEIm2: 25576 T: 134,01 D: 0 PEIm1: 25576 PEIm2: 25576 T: 124,924 D: 0 PEIm1: 25576 PEIm2: 25576 T: 125,114 D: 0 PEIm1: 25576 PEIm2: 25576 T: 125 D: 0 PEIm1: 25576 PEIm2: 25576 T: 127,632 D: 0 PEIm1: 25576 PEIm2: 25576 — 27k — T: 149,386 D: 0 PEIm1: 27927 PEIm2: 27927 T: 149,576 D: 0 PEIm1: 27927 PEIm2: 27927 T: 149,529 D: 0 PEIm1: 27927 PEIm2: 27927 T: 149,572 D: 0 PEIm1: 27927 PEIm2: 27927 T: 156,489 D: 0 PEIm1: 27927 PEIm2: 27927 T: 187,503 D: 0 PEIm1: 27927 PEIm2: 27927 T: 156,149 D: 0 PEIm1: 27927 PEIm2: 27927 T: 149,6 D: 0 PEIm1: 27927 PEIm2: 27927 T: 149,312 D: 0 PEIm1: 27927 PEIm2: 27927 T: 150,032 D: 0 PEIm1: 27927 PEIm2: 27927 — 30k — T: 189,13 D: 0 PEIm1: 30337 PEIm2: 30337 T: 176,66 D: 0 PEIm1: 30337 PEIm2: 30337 T: 176,702 D: 0 PEIm1: 30337 PEIm2: 30337 T: 176,579 D: 0 PEIm1: 30337 PEIm2: 30337 T: 175,11 D: 0 PEIm1: 30337 PEIm2: 30337 T: 176,66 D: 0 PEIm1: 30337 PEIm2: 30337 T: 180,16 D: 0 PEIm1: 30337 PEIm2: 30337 T: 176,633 D: 0 PEIm1: 30337 PEIm2: 30337 T: 180,197 D: 0 PEIm1: 30337 PEIm2: 30337 T: 176,681 D: 0 PEIm1: 30337 PEIm2: 30337 — 41k — T: 330,458 D: 0 PEIm1: 41533 PEIm2: 41533 T: 330,371 D: 0 PEIm1: 41533 PEIm2: 41533 T: 330,416 D: 0 PEIm1: 41533 PEIm2: 41533 T: 328,704 D: 0 PEIm1: 41533 PEIm2: 41533 T: 330,274 D: 0 PEIm1: 41533 PEIm2: 41533 T: 343,001 D: 0 PEIm1: 41533 PEIm2: 41533 T: 344,744 D: 0 PEIm1: 41533 PEIm2: 41533 T: 346,222 D: 0 PEIm1: 41533 PEIm2: 41533 T: 330,399 D: 0 PEIm1: 41533 PEIm2: 41533 T: 330,517 D: 0 PEIm1: 41533 PEIm2: 41533 — 47k — T: 427,148 D: 0 PEIm1: 47141 PEIm2: 47141 T: 427,397 D: 0 PEIm1: 47141 PEIm2: 47141 T: 425,967 D: 0 PEIm1: 47141 PEIm2: 47141 T: 426,842 D: 0 PEIm1: 47141 PEIm2: 47141 T: 426,703 D: 0 PEIm1: 47141 PEIm2: 47141 T: 427,025 D: 0 PEIm1: 47141 PEIm2: 47141 T: 426,846 D: 0 PEIm1: 47141 PEIm2: 47141 T: 456,907 D: 0 PEIm1: 47141 PEIm2: 47141 T: 426,972 D: 0 PEIm1: 47141 PEIm2: 47141 T: 449,672 D: 0 PEIm1: 47141 PEIm2: 47141 — 54k — T: 576,811 D: 0 PEIm1: 54895 PEIm2: 54895 T: 577,04 D: 0 PEIm1: 54895 PEIm2: 54895 T: 576,844 D: 0 PEIm1: 54895 PEIm2: 54895 T: 576,861 D: 0 PEIm1: 54895 PEIm2: 54895 T: 608,239 D: 0 PEIm1: 54895 PEIm2: 54895 T: 576,616 D: 0 PEIm1: 54895 PEIm2: 54895 T: 576,862 D: 0 PEIm1: 54895 PEIm2: 54895 T: 576,231 D: 0 PEIm1: 54895 PEIm2: 54895 T: 588,612 D: 0 PEIm1: 54895 PEIm2: 54895 T: 576,993 D: 0 PEIm1: 54895 PEIm2: 54895 — 57k — T: 671,835 D: 0 PEIm1: 57726 PEIm2: 57726 T: 637,601 D: 0 PEIm1: 57726 PEIm2: 57726 T: 665,375 D: 0 PEIm1: 57726 PEIm2: 57726 T: 650,499 D: 0 PEIm1: 57726 PEIm2: 57726 T: 631,981 D: 0 PEIm1: 57726 PEIm2: 57726 T: 637,154 D: 0 PEIm1: 57726 PEIm2: 57726 T: 637,275 D: 0 PEIm1: 57726 PEIm2: 57726 T: 668,695 D: 0 PEIm1: 57726 PEIm2: 57726 T: 652,319 D: 0 PEIm1: 57726 PEIm2: 57726 T: 631,963 D: 0 PEIm1: 57726 PEIm2: 57726 — 68k — T: 912,045 D: 0 PEIm1: 68761 PEIm2: 68761 T: 901,118 D: 0 PEIm1: 68761 PEIm2: 68761 T: 902,373 D: 0 PEIm1: 68761 PEIm2: 68761 T: 903,086 D: 0 PEIm1: 68761 PEIm2: 68761 T: 944,792 D: 0 PEIm1: 68761 PEIm2: 68761 T: 902,404 D: 0 PEIm1: 68761 PEIm2: 68761 T: 1310,78 D: 0 PEIm1: 68761 PEIm2: 68761 T: 902,916 D: 0 PEIm1: 68761 PEIm2: 68761 T: 902,898 D: 0 PEIm1: 68761 PEIm2: 68761 T: 945,508 D: 0 PEIm1: 68761 PEIm2: 68761 — 84k — T: 1348,31 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1348,64 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1347,85 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1347,79 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1443,31 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1348,11 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1348,13 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1348,08 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1420,4 D: 0 PEIm1: 83942 PEIm2: 83942 T: 1348,08 D: 0 PEIm1: 83942 PEIm2: 83942 — 108k — T: 2228,72 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2227,85 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2350,27 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2228,77 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2273,46 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2229,43 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2227,27 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2228,18 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2350,12 D: 0 PEIm1: 108076 PEIm2: 108076 T: 2228,81 D: 0 PEIm1: 108076 PEIm2: 108076 — 123k —

B.3.2. 2 Hilos

— 5k — T: 5,2345 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,11383 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,47995

D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,11398 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,26493 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,11464 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,20362 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,11378 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,4801 D: 0 PEIm1: 7219 PEIm2: 7219 T: 5,11362 D: 0 PEIm1: 7219 PEIm2: 7219 — 9k — T: 12,0055 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,0062 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,006 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,0058 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,0056 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,0057 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,1229 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,0057 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,0138 D: 0 PEIm1: 11198 PEIm2: 11198 T: 12,0051 D: 0 PEIm1: 11198 PEIm2: 11198 — 11k — T: 21,8574 D: 0 PEIm1: 15040 PEIm2: 15040 T: 21,8579 D: 0 PEIm1: 15040 PEIm2: 15040 T: 22,2868 D: 0 PEIm1: 15040 PEIm2: 15040 T: 21,9963 D: 0 PEIm1: 15040 PEIm2: 15040 T: 22,2143 D: 0 PEIm1: 15040 PEIm2: 15040 T: 22,351 D: 0 PEIm1: 15040 PEIm2: 15040 T: 21,8579 D: 0 PEIm1: 15040 PEIm2: 15040 T: 22,2229 D: 0 PEIm1: 15040 PEIm2: 15040 T: 21,8583 D: 0 PEIm1: 15040 PEIm2: 15040 T: 21,859 D: 0 PEIm1: 15040 PEIm2: 15040 — 15k — T: 39,2387 D: 0 PEIm1: 20189 PEIm2: 20189 T: 39,2385 D: 0 PEIm1: 20189 PEIm2: 20189 T: 39,2391 D: 0 PEIm1: 20189 PEIm2: 20189 T: 40,1373 D: 0 PEIm1: 20189 PEIm2: 20189 T: 39,2369 D: 0 PEIm1: 20189 PEIm2: 20189 T: 39,2472 D: 0 PEIm1: 20189 PEIm2: 20189 T: 39,8771 D: 0 PEIm1: 20189 PEIm2: 20189 T: 39,237 D: 0 PEIm1: 20189 PEIm2: 20189 T: 42,1588 D: 0 PEIm1: 20189 PEIm2: 20189 T: 39,2377 D: 0 PEIm1: 20189 PEIm2: 20189 — 19k — T: 36,6936 D: 0 PEIm1: 19490 PEIm2: 19490 T: 36,6866 D: 0 PEIm1: 19490 PEIm2: 19490 T: 36,6859 D: 0 PEIm1: 19490 PEIm2: 19490 T: 36,6877 D: 0 PEIm1: 19490 PEIm2: 19490 T: 36,688 D: 0 PEIm1: 19490 PEIm2: 19490 T: 36,9184 D: 0 PEIm1: 19490 PEIm2: 19490 T: 36,6864 D: 0 PEIm1: 19490 PEIm2: 19490 T: 39,4022 D: 0 PEIm1: 19490 PEIm2: 19490 T: 36,6874 D: 0 PEIm1: 19490 PEIm2: 19490 T: 39,402 D: 0 PEIm1: 19490 PEIm2: 19490 — 25k — T: 66,0158 D: 0 PEIm1: 25576 PEIm2: 25576 T: 63,5657 D: 0 PEIm1: 25576 PEIm2: 25576 T: 62,3535 D: 0 PEIm1: 25576 PEIm2: 25576 T: 62,3544 D: 0 PEIm1: 25576 PEIm2: 25576 T: 62,3584 D: 0 PEIm1: 25576 PEIm2: 25576 T: 67,1437 D: 0 PEIm1: 25576 PEIm2: 25576 T: 63,5933 D: 0 PEIm1: 25576 PEIm2: 25576 T: 62,354 D: 0 PEIm1: 25576 PEIm2: 25576 T: 66,02 D: 0 PEIm1: 25576 PEIm2: 25576 T: 62,7552 D: 0 PEIm1: 25576 PEIm2: 25576 — 27k — T: 74,6213 D: 0 PEIm1: 27927 PEIm2: 27927 T: 79,066 D: 0 PEIm1: 27927 PEIm2: 27927 T: 80,3108 D: 0 PEIm1: 27927 PEIm2: 27927 T: 76,0796 D: 0 PEIm1: 27927 PEIm2: 27927 T: 76,4334 D: 0 PEIm1: 27927 PEIm2: 27927 T: 74,6269 D: 0 PEIm1: 27927 PEIm2: 27927 T: 74,6193 D: 0 PEIm1: 27927 PEIm2: 27927 T: 74,6177 D: 0 PEIm1: 27927 PEIm2: 27927 T: 79,0722 D: 0 PEIm1: 27927 PEIm2: 27927 T: 74,6195 D: 0 PEIm1: 27927 PEIm2: 27927 — 30k — T: 88,3258 D: 0 PEIm1: 30337 PEIm2: 30337 T: 93,4645 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,3318 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,3397 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,3284 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,3267 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,3252 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,3355 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,325 D: 0 PEIm1: 30337 PEIm2: 30337 T: 88,3267 D: 0 PEIm1: 30337 PEIm2: 30337 — 41k — T: 168,148 D: 0 PEIm1: 41533 PEIm2: 41533 T: 164,78 D: 0 PEIm1: 41533 PEIm2: 41533 T: 177,347 D: 0 PEIm1: 41533 PEIm2: 41533 T: 174,606 D: 0 PEIm1: 41533 PEIm2: 41533 T: 164,777 D: 0 PEIm1: 41533 PEIm2: 41533 T: 168,69 D: 0 PEIm1: 41533 PEIm2: 41533 T: 164,806 D: 0 PEIm1: 41533 PEIm2: 41533 T: 164,794 D: 0 PEIm1: 41533 PEIm2: 41533 T: 164,791 D: 0 PEIm1: 41533 PEIm2: 41533 T: 164,802 D: 0 PEIm1: 41533 PEIm2: 41533 — 47k — T: 213,085 D: 0 PEIm1: 47141 PEIm2: 47141 T: 229,196 D: 0 PEIm1: 47141 PEIm2: 47141 T: 213,104 D: 0 PEIm1: 47141 PEIm2: 47141 T: 229,411 D: 0 PEIm1: 47141 PEIm2: 47141 T: 219,07 D: 0 PEIm1: 47141 PEIm2: 47141 T: 213,096 D: 0 PEIm1: 47141 PEIm2: 47141 T: 218,092 D: 0 PEIm1: 47141 PEIm2: 47141 T: 213,077 D: 0 PEIm1: 47141 PEIm2: 47141 T: 225,342 D: 0 PEIm1: 47141 PEIm2: 47141 T: 213,08 D: 0 PEIm1: 47141 PEIm2: 47141 — 54k — T: 287,783 D: 0 PEIm1: 54895 PEIm2: 54895 T: 287,792 D: 0 PEIm1: 54895 PEIm2: 54895 T: 287,787 D: 0 PEIm1: 54895 PEIm2: 54895 T: 305,023 D: 0 PEIm1: 54895 PEIm2: 54895 T: 289,089 D: 0 PEIm1: 54895 PEIm2: 54895 T: 289,162 D: 0 PEIm1: 54895 PEIm2: 54895 T: 289,157 D: 0 PEIm1: 54895 PEIm2: 54895 T: 289,138 D: 0 PEIm1: 54895 PEIm2: 54895 T: 289,156 D: 0 PEIm1: 54895 PEIm2: 54895 T: 289,15 D: 0 PEIm1: 54895 PEIm2: 54895 — 57k — T: 326,986 D: 0 PEIm1: 57726 PEIm2: 57726 T: 325,019 D: 0 PEIm1: 57726 PEIm2: 57726 T: 319,41 D: 0 PEIm1: 57726 PEIm2: 57726 T: 319,415 D: 0 PEIm1: 57726 PEIm2: 57726 T: 319,412 D: 0 PEIm1: 57726 PEIm2: 57726 T: 337,961 D: 0 PEIm1: 57726 PEIm2: 57726 T: 328,621 D: 0 PEIm1: 57726 PEIm2: 57726 T: 319,399 D: 0 PEIm1: 57726 PEIm2: 57726 T: 319,433 D: 0 PEIm1: 57726 PEIm2: 57726 T: 319,484 D: 0 PEIm1: 57726 PEIm2: 57726 — 68k — T: 451,646 D: 0 PEIm1: 68761 PEIm2: 68761 T: 451,748 D: 0 PEIm1: 68761 PEIm2: 68761 T: 460,565 D: 0 PEIm1: 68761 PEIm2: 68761 T: 459,374 D: 0 PEIm1: 68761 PEIm2: 68761 T: 477,979 D: 0 PEIm1: 68761 PEIm2: 68761 T: 449,471 D: 0 PEIm1: 68761 PEIm2: 68761 T: 452,682 D: 0 PEIm1: 68761 PEIm2: 68761 T: 450,2 D: 0 PEIm1: 68761 PEIm2: 68761 T: 452,352 D: 0 PEIm1: 68761 PEIm2: 68761 T: 449,45 D: 0 PEIm1: 68761 PEIm2: 68761 — 84k — T: 671,772 D: 0 PEIm1: 83942 PEIm2: 83942 T: 671,701 D: 0 PEIm1: 83942 PEIm2: 83942 T: 671,769 D: 0 PEIm1: 83942 PEIm2: 83942 T: 671,709 D: 0 PEIm1: 83942 PEIm2: 83942 T: 671,704 D: 0 PEIm1: 83942 PEIm2: 83942 T: 671,691 D: 0 PEIm1: 83942 PEIm2: 83942 T: 671,759 D: 0 PEIm1: 83942 PEIm2: 83942 T: 671,715 D: 0 PEIm1: 83942 PEIm2: 83942 T: 672,537 D: 0 PEIm1: 83942 PEIm2: 83942 T: 723,302 D: 0 PEIm1: 83942 PEIm2: 83942 — 108k — T: 1109,09 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1109,14 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1109,08 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1109,13 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1109,09 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1109,11 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1174,29 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1109,1 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1121,34 D: 0 PEIm1: 108076 PEIm2: 108076 T: 1124,71 D: 0 PEIm1: 108076 PEIm2: 108076 — 123k —

B.3.3. 4 Hilos

— 5k — T: 2,61633 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,55654 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,56557 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,5685 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,55678 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,61717 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,5659 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,56961 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,56576 D: 0 PEIm1: 7219 PEIm2: 7219 T: 2,56584 D: 0 PEIm1: 7219 PEIm2: 7219 — 9k — T: 6,12687 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,03534 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,14188 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,03484 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,03427 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,03372 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,13713 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,14256 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,03479 D: 0 PEIm1: 11198 PEIm2: 11198 T: 6,03349 D: 0 PEIm1: 11198 PEIm2: 11198 — 11k — T: 10,993 D: 0 PEIm1: 15040 PEIm2: 15040 T: 11,1849 D: 0 PEIm1: 15040 PEIm2: 15040 T: 10,9931 D: 0 PEIm1: 15040 PEIm2: 15040 T: 10,9922 D: 0 PEIm1: 15040 PEIm2: 15040 T: 10,993 D: 0 PEIm1: 15040 PEIm2: 15040 T: 11,797 D: 0 PEIm1: 15040 PEIm2: 15040 T: 11,6259 D: 0 PEIm1: 15040 PEIm2: 15040 T: 10,9957 D: 0 PEIm1: 15040 PEIm2: 15040 T: 11,0596 D: 0 PEIm1: 15040 PEIm2: 15040 T: 10,9954 D: 0 PEIm1: 15040 PEIm2: 15040 — 15k — T: 19,7115 D: 0 PEIm1: 20189 PEIm2: 20189 T: 19,7081 D: 0 PEIm1: 20189 PEIm2: 20189 T: 20,8274 D: 0 PEIm1: 20189 PEIm2: 20189 T: 19,7079 D: 0 PEIm1: 20189 PEIm2: 20189 T: 20,2176 D: 0 PEIm1: 20189 PEIm2: 20189 T: 19,7081 D: 0 PEIm1: 20189 PEIm2: 20189 T: 20,8455 D: 0 PEIm1: 20189 PEIm2: 20189 T: 19,7074 D: 0 PEIm1: 20189 PEIm2: 20189 T: 19,707 D: 0 PEIm1: 20189 PEIm2: 20189 T: 20,2403 D: 0 PEIm1: 20189 PEIm2: 20189 — 19k — T: 18,408 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,4073 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,4074 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,4076 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,5194 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,4087 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,408 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,4077 D: 0 PEIm1: 19490 PEIm2: 19490 T: 19,4669 D: 0 PEIm1: 19490 PEIm2: 19490 T: 18,4074 D: 0 PEIm1: 19490 PEIm2: 19490 — 25k — T: 31,2653 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,271 D: 0 PEIm1: 25576 PEIm2: 25576 T: 33,1568 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,2686 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,2702 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,2678 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,2642 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,2651 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,2652 D: 0 PEIm1: 25576 PEIm2: 25576 T: 31,2689 D: 0 PEIm1: 25576 PEIm2: 25576 — 27k — T: 37,4135 D: 0 PEIm1: 27927 PEIm2: 27927 T: 37,4123 D: 0 PEIm1: 27927 PEIm2: 27927 T: 38,3134 D: 0 PEIm1: 27927 PEIm2: 27927 T: 38,0784 D: 0 PEIm1: 27927 PEIm2: 27927 T: 38,0906 D: 0 PEIm1: 27927 PEIm2: 27927 T: 37,4134 D: 0 PEIm1: 27927 PEIm2: 27927 T: 37,4088 D: 0 PEIm1: 27927 PEIm2: 27927 T: 37,5548 D: 0 PEIm1: 27927 PEIm2: 27927 T: 37,4092 D: 0 PEIm1: 27927 PEIm2: 27927 T: 38,077 D: 0 PEIm1: 27927 PEIm2: 27927 — 30k — T: 44,3267 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,3229 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,2547 D: 0 PEIm1: 30337 PEIm2: 30337 T: 46,8407 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,2563 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,2562 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,2576 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,2733 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,3185 D: 0 PEIm1: 30337 PEIm2: 30337 T: 44,3171 D: 0 PEIm1: 30337 PEIm2: 30337 — 41k — T: 82,5757 D: 0 PEIm1: 41533 PEIm2: 41533 T: 82,5826 D: 0 PEIm1: 41533 PEIm2: 41533 T: 82,5817 D: 0 PEIm1: 41533 PEIm2: 41533 T: 84,667 D: 0 PEIm1: 41533 PEIm2: 41533 T: 82,587 D: 0 PEIm1: 41533 PEIm2: 41533 T: 82,5791 D: 0 PEIm1: 41533 PEIm2: 41533 T: 84,0193 D: 0 PEIm1: 41533 PEIm2: 41533 T: 88,3583 D: 0 PEIm1: 41533 PEIm2: 41533 T: 87,3796 D:

0 PEIm1: 41533 PEIm2: 41533 T: 82,582 D: 0 PEIm1: 41533 PEIm2: 41533 — 47k — T: 108,5 D: 0 PEIm1: 47141 PEIm2: 47141 T: 106,688 D: 0 PEIm1: 47141 PEIm2: 47141 T: 108,432 D: 0 PEIm1: 47141 PEIm2: 47141 T: 106,686 D: 0 PEIm1: 47141 PEIm2: 47141 T: 106,686 D: 0 PEIm1: 47141 PEIm2: 47141 T: 106,716 D: 0 PEIm1: 47141 PEIm2: 47141 T: 106,906 D: 0 PEIm1: 47141 PEIm2: 47141 T: 106,913 D: 0 PEIm1: 47141 PEIm2: 47141 T: 106,796 D: 0 PEIm1: 47141 PEIm2: 47141 T: 114,726 D: 0 PEIm1: 47141 PEIm2: 47141 — 54k — T: 144,188 D: 0 PEIm1: 54895 PEIm2: 54895 T: 146,634 D: 0 PEIm1: 54895 PEIm2: 54895 T: 144,173 D: 0 PEIm1: 54895 PEIm2: 54895 T: 144,173 D: 0 PEIm1: 54895 PEIm2: 54895 T: 152,587 D: 0 PEIm1: 54895 PEIm2: 54895 T: 144,166 D: 0 PEIm1: 54895 PEIm2: 54895 T: 144,173 D: 0 PEIm1: 54895 PEIm2: 54895 T: 144,248 D: 0 PEIm1: 54895 PEIm2: 54895 T: 144,348 D: 0 PEIm1: 54895 PEIm2: 54895 T: 144,347 D: 0 PEIm1: 54895 PEIm2: 54895 — 57k — T: 159,372 D: 0 PEIm1: 57726 PEIm2: 57726 T: 159,36 D: 0 PEIm1: 57726 PEIm2: 57726 T: 159,362 D: 0 PEIm1: 57726 PEIm2: 57726 T: 160,27 D: 0 PEIm1: 57726 PEIm2: 57726 T: 171,582 D: 0 PEIm1: 57726 PEIm2: 57726 T: 159,383 D: 0 PEIm1: 57726 PEIm2: 57726 T: 171,572 D: 0 PEIm1: 57726 PEIm2: 57726 T: 159,345 D: 0 PEIm1: 57726 PEIm2: 57726 T: 163,129 D: 0 PEIm1: 57726 PEIm2: 57726 T: 162,408 D: 0 PEIm1: 57726 PEIm2: 57726 — 68k — T: 225,662 D: 0 PEIm1: 68761 PEIm2: 68761 T: 227,289 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,992 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,687 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,693 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,673 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,72 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,734 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,683 D: 0 PEIm1: 68761 PEIm2: 68761 T: 225,658 D: 0 PEIm1: 68761 PEIm2: 68761 — 84k — T: 357,227 D: 0 PEIm1: 83942 PEIm2: 83942 T: 336,534 D: 0 PEIm1: 83942 PEIm2: 83942 T: 336,508 D: 0 PEIm1: 83942 PEIm2: 83942 T: 346,183 D: 0 PEIm1: 83942 PEIm2: 83942 T: 336,501 D: 0 PEIm1: 83942 PEIm2: 83942 T: 342,422 D: 0 PEIm1: 83942 PEIm2: 83942 T: 336,494 D: 0 PEIm1: 83942 PEIm2: 83942 T: 336,502 D: 0 PEIm1: 83942 PEIm2: 83942 T: 336,492 D: 0 PEIm1: 83942 PEIm2: 83942 T: 336,487 D: 0 PEIm1: 83942 PEIm2: 83942 — 108k — T: 588,938 D: 0 PEIm1: 108076 PEIm2: 108076 T: 555,479 D: 0 PEIm1: 108076 PEIm2: 108076 T: 559,82 D: 0 PEIm1: 108076 PEIm2: 108076 T: 556,553 D: 0 PEIm1: 108076 PEIm2: 108076 T: 590,436 D: 0 PEIm1: 108076 PEIm2: 108076 T: 572,897 D: 0 PEIm1: 108076 PEIm2: 108076 T: 556,64 D: 0 PEIm1: 108076 PEIm2: 108076 T: 569,813 D: 0 PEIm1: 108076 PEIm2: 108076 T: 556,556 D: 0 PEIm1: 108076 PEIm2: 108076 T: 556,531 D: 0 PEIm1: 108076 PEIm2: 108076 — 123k —

B.3.4. 8 Hilos

— 5k — T: 1,29112 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,28266 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,35152 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,28356 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,28263 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,28286 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,28345 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,28682 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,32309 D: 0 PEIm1: 7219 PEIm2: 7219 T: 1,28292 D: 0 PEIm1: 7219 PEIm2: 7219 — 9k — T: 3,02586 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02598 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02562 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02495 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02598 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02595 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02544 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02512 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,02539 D: 0 PEIm1: 11198 PEIm2: 11198 T: 3,04405 D: 0 PEIm1: 11198 PEIm2: 11198 — 11k — T: 5,51125 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,51309 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,51327 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,82775 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,51319 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,81829 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,49975 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,51089 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,5112 D: 0 PEIm1: 15040 PEIm2: 15040 T: 5,51178 D: 0 PEIm1: 15040 PEIm2: 15040 — 15k — T: 10,0076 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,8538 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,87768 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,87636 D: 0 PEIm1: 20189 PEIm2: 20189 T: 10,5962 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,87622 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,87722 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,8778 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,87725 D: 0 PEIm1: 20189 PEIm2: 20189 T: 9,87702 D: 0 PEIm1: 20189 PEIm2: 20189 — 19k — T: 9,20126 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,19704 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,35471 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,1994 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,20149 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,71786 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,2003 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,20067 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,19919 D: 0 PEIm1: 19490 PEIm2: 19490 T: 9,40332 D: 0 PEIm1: 19490 PEIm2: 19490 — 25k — T: 15,6473 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,6496 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,646 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,6506 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,6452 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,9105 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,6483 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,9217 D: 0 PEIm1: 25576 PEIm2: 25576 T: 16,837 D: 0 PEIm1: 25576 PEIm2: 25576 T: 15,6486 D: 0 PEIm1: 25576 PEIm2: 25576 — 27k — T: 18,8599 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,7339 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,7363 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,7332 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,9193 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,7368 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,7274 D: 0 PEIm1: 27927 PEIm2: 27927 T: 19,8342 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,7155 D: 0 PEIm1: 27927 PEIm2: 27927 T: 18,7331 D: 0 PEIm1: 27927 PEIm2: 27927 — 30k — T: 22,1773 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,176 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,3172 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,5724 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,1757 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,1757 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,1801 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,2017 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,5619 D: 0 PEIm1: 30337 PEIm2: 30337 T: 22,1784 D: 0 PEIm1: 30337 PEIm2: 30337 — 41k — T: 41,4192 D: 0 PEIm1: 41533 PEIm2: 41533 T: 41,2951 D: 0 PEIm1: 41533 PEIm2: 41533 T: 41,4178 D: 0 PEIm1: 41533 PEIm2: 41533 T: 41,2928 D: 0 PEIm1: 41533 PEIm2: 41533 T: 42,3937 D: 0 PEIm1: 41533 PEIm2: 41533 T: 41,2961 D: 0 PEIm1: 41533 PEIm2: 41533 T: 43,8714 D: 0 PEIm1: 41533 PEIm2: 41533 T: 41,2964 D: 0 PEIm1: 41533 PEIm2: 41533 T: 41,4179 D: 0 PEIm1: 41533 PEIm2: 41533 T: 41,292 D: 0 PEIm1: 41533 PEIm2: 41533 — 47k — T: 53,4864 D: 0 PEIm1: 47141 PEIm2: 47141 T: 53,4382 D: 0 PEIm1: 47141 PEIm2: 47141 T: 57,5107 D: 0 PEIm1: 47141 PEIm2: 47141 T: 56,5454 D: 0 PEIm1: 47141 PEIm2: 47141 T: 54,4543 D: 0 PEIm1: 47141 PEIm2: 47141 T: 54,6838 D: 0 PEIm1: 47141 PEIm2: 47141 T: 57,5113 D: 0 PEIm1: 47141 PEIm2: 47141 T: 53,4371 D: 0 PEIm1: 47141 PEIm2: 47141 T: 53,4908 D: 0 PEIm1: 47141 PEIm2: 47141 T: 53,4385 D: 0 PEIm1: 47141 PEIm2: 47141 — 54k — T: 77,6508 D: 0 PEIm1: 54895 PEIm2: 54895 T: 72,2566 D: 0 PEIm1: 54895 PEIm2: 54895 T: 72,2639 D: 0 PEIm1: 54895 PEIm2: 54895 T: 72,2558 D: 0 PEIm1: 54895 PEIm2: 54895 T: 72,2601 D: 0 PEIm1: 54895 PEIm2: 54895 T: 74,335 D: 0 PEIm1: 54895 PEIm2: 54895 T: 72,4085 D: 0 PEIm1: 54895 PEIm2: 54895 T: 72,2564 D: 0 PEIm1: 54895 PEIm2: 54895 T: 73,6047 D: 0 PEIm1: 54895 PEIm2: 54895 T: 72,264 D: 0 PEIm1: 54895 PEIm2: 54895 — 57k — T: 84,5925 D: 0 PEIm1: 57726 PEIm2: 57726 T: 79,8069 D: 0 PEIm1: 57726 PEIm2: 57726 T: 79,8082 D: 0 PEIm1: 57726 PEIm2: 57726 T: 79,8082 D: 0 PEIm1: 57726 PEIm2: 57726 T: 84,5982 D: 0 PEIm1: 57726 PEIm2: 57726 T: 79,8084 D: 0 PEIm1: 57726 PEIm2: 57726 T: 79,8072 D: 0 PEIm1: 57726 PEIm2: 57726 T: 79,8106 D: 0 PEIm1: 57726 PEIm2: 57726 T: 81,2059 D: 0 PEIm1: 57726 PEIm2: 57726 T: 79,8072 D: 0 PEIm1: 57726 PEIm2: 57726 — 68k — T: 112,793 D: 0 PEIm1: 68761 PEIm2: 68761 T: 112,811 D: 0 PEIm1: 68761 PEIm2: 68761 T: 112,79 D: 0 PEIm1: 68761 PEIm2: 68761 T: 114,766 D: 0 PEIm1: 68761 PEIm2: 68761 T: 119,393 D: 0 PEIm1: 68761 PEIm2: 68761 T: 112,787 D: 0 PEIm1: 68761 PEIm2: 68761 T: 112,794 D: 0 PEIm1: 68761 PEIm2: 68761 T: 112,792 D: 0 PEIm1: 68761 PEIm2: 68761 T: 112,789 D: 0 PEIm1: 68761 PEIm2: 68761 T: 112,789 D: 0 PEIm1: 68761 PEIm2: 68761 — 84k — T: 168,619 D: 0 PEIm1: 83942 PEIm2: 83942 T: 168,628 D: 0 PEIm1: 83942 PEIm2: 83942 T: 168,624 D: 0 PEIm1: 83942 PEIm2: 83942 T: 168,62 D: 0 PEIm1: 83942 PEIm2: 83942 T: 173,497 D: 0 PEIm1: 83942 PEIm2: 83942 T: 168,63 D: 0 PEIm1: 83942 PEIm2: 83942 T: 168,626 D: 0 PEIm1: 83942 PEIm2: 83942 T: 178,758 D: 0 PEIm1: 83942 PEIm2: 83942 T: 168,624 D: 0 PEIm1: 83942 PEIm2: 83942 T: 168,628 D: 0 PEIm1: 83942 PEIm2: 83942 — 108k — T: 278,393 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,368 D: 0 PEIm1: 108076 PEIm2: 108076 T: 280,765 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,366 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,409 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,368 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,383 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,41 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,372 D: 0 PEIm1: 108076 PEIm2: 108076 T: 278,373 D: 0 PEIm1: 108076 PEIm2: 108076 — 123k —

B.3.5. 12 Hilos

— 5k — T: 0,882446 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,855674 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,857799 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,858926 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,856627 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,904993 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,857926 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,859768 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,85678 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,855786 D: 0 PEIm1: 7219 PEIm2: 7219 — 9k — T: 2,13745 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,17609 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,05183 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,06338 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,01765 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,02543 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,07586 D: 0 PEIm1: 11198 PEIm2:

11198 T: 2,09506 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,05327 D: 0 PEIm1: 11198 PEIm2: 11198 T: 2,02664 D: 0 PEIm1: 11198 PEIm2: 11198 — 11k — T: 3,67032 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,76049 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,67978 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,89937 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,67929 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,67963 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,67942 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,67072 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,67931 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,6791 D: 0 PEIm1: 15040 PEIm2: 15040 — 15k — T: 6,59884 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,60489 D: 0 PEIm1: 20189 PEIm2: 20189 T: 7,06048 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,75517 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,95629 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,57744 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,70142 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,57966 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,57718 D: 0 PEIm1: 20189 PEIm2: 20189 T: 6,75532 D: 0 PEIm1: 20189 PEIm2: 20189 — 19k — T: 6,1749 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,2798 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,22885 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,49607 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,59318 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,14417 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,13834 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,30861 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,14334 D: 0 PEIm1: 19490 PEIm2: 19490 T: 6,16102 D: 0 PEIm1: 19490 PEIm2: 19490 — 25k — T: 10,4152 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4228 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4229 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4228 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4126 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4227 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4228 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4228 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4228 D: 0 PEIm1: 25576 PEIm2: 25576 T: 10,4237 D: 0 PEIm1: 25576 PEIm2: 25576 — 27k — T: 12,4652 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,4804 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,4768 D: 0 PEIm1: 27927 PEIm2: 27927 T: 13,4101 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,4767 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,4651 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,6835 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,683 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,4768 D: 0 PEIm1: 27927 PEIm2: 27927 T: 12,4771 D: 0 PEIm1: 27927 PEIm2: 27927 — 30k — T: 14,7655 D: 0 PEIm1: 30337 PEIm2: 30337 T: 14,7962 D: 0 PEIm1: 30337 PEIm2: 30337 T: 14,7965 D: 0 PEIm1: 30337 PEIm2: 30337 T: 14,7955 D: 0 PEIm1: 30337 PEIm2: 30337 T: 14,8818 D: 0 PEIm1: 30337 PEIm2: 30337 T: 14,7972 D: 0 PEIm1: 30337 PEIm2: 30337 T: 14,7957 D: 0 PEIm1: 30337 PEIm2: 30337 T: 15,2154 D: 0 PEIm1: 30337 PEIm2: 30337 T: 14,8833 D: 0 PEIm1: 30337 PEIm2: 30337 T: 15,6507 D: 0 PEIm1: 30337 PEIm2: 30337 — 41k — T: 27,5378 D: 0 PEIm1: 41533 PEIm2: 41533 T: 28,0698 D: 0 PEIm1: 41533 PEIm2: 41533 T: 29,6273 D: 0 PEIm1: 41533 PEIm2: 41533 T: 27,564 D: 0 PEIm1: 41533 PEIm2: 41533 T: 27,5637 D: 0 PEIm1: 41533 PEIm2: 41533 T: 27,5211 D: 0 PEIm1: 41533 PEIm2: 41533 T: 29,6518 D: 0 PEIm1: 41533 PEIm2: 41533 T: 27,5784 D: 0 PEIm1: 41533 PEIm2: 41533 T: 28,0227 D: 0 PEIm1: 41533 PEIm2: 41533 T: 27,5644 D: 0 PEIm1: 41533 PEIm2: 41533 — 47k — T: 35,571 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,5723 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,7025 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,5739 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,5932 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,723 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,5607 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,9228 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,5944 D: 0 PEIm1: 47141 PEIm2: 47141 T: 35,5725 D: 0 PEIm1: 47141 PEIm2: 47141 — 54k — T: 48,1036 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,1021 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,3185 D: 0 PEIm1: 54895 PEIm2: 54895 T: 49,2456 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,0923 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,1035 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,0922 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,1018 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,0929 D: 0 PEIm1: 54895 PEIm2: 54895 T: 48,1064 D: 0 PEIm1: 54895 PEIm2: 54895 — 57k — T: 53,1786 D: 0 PEIm1: 57726 PEIm2: 57726 T: 53,1784 D: 0 PEIm1: 57726 PEIm2: 57726 T: 53,1734 D: 0 PEIm1: 57726 PEIm2: 57726 T: 56,3278 D: 0 PEIm1: 57726 PEIm2: 57726 T: 53,1741 D: 0 PEIm1: 57726 PEIm2: 57726 T: 53,18 D: 0 PEIm1: 57726 PEIm2: 57726 T: 53,1962 D: 0 PEIm1: 57726 PEIm2: 57726 T: 57,2074 D: 0 PEIm1: 57726 PEIm2: 57726 T: 53,5372 D: 0 PEIm1: 57726 PEIm2: 57726 T: 54,4369 D: 0 PEIm1: 57726 PEIm2: 57726 — 68k — T: 76,3451 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,0038 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,0042 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,0059 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,1544 D: 0 PEIm1: 68761 PEIm2: 68761 T: 76,3773 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,0028 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,0069 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,0055 D: 0 PEIm1: 68761 PEIm2: 68761 T: 75,0106 D: 0 PEIm1: 68761 PEIm2: 68761 — 84k — T: 112,214 D: 0 PEIm1: 83942 PEIm2: 83942 T: 112,221 D: 0 PEIm1: 83942 PEIm2: 83942 T: 114,877 D: 0 PEIm1: 83942 PEIm2: 83942 T: 112,22 D: 0 PEIm1: 83942 PEIm2: 83942 T: 112,221 D: 0 PEIm1: 83942 PEIm2: 83942 T: 112,133 D: 0 PEIm1: 83942 PEIm2: 83942 T: 112,186 D: 0 PEIm1: 83942 PEIm2: 83942 T: 118,921 D: 0 PEIm1: 83942 PEIm2: 83942 T: 112,22 D: 0 PEIm1: 83942 PEIm2: 83942 T: 112,221 D: 0 PEIm1: 83942 PEIm2: 83942 — 108k — T: 185,348 D: 0 PEIm1: 108076 PEIm2: 108076 T: 187,605 D: 0 PEIm1: 108076 PEIm2: 108076 T: 185,37 D: 0 PEIm1: 108076 PEIm2: 108076 T: 185,348 D: 0 PEIm1: 108076 PEIm2: 108076 T: 185,349 D: 0 PEIm1: 108076 PEIm2: 108076 T: 189,37 D: 0 PEIm1: 108076 PEIm2: 108076 T: 185,581 D: 0 PEIm1: 108076 PEIm2: 108076 T: 185,364 D: 0 PEIm1: 108076 PEIm2: 108076 T: 199,656 D: 0 PEIm1: 108076 PEIm2: 108076 T: 185,35 D: 0 PEIm1: 108076 PEIm2: 108076 — 123k — T: 243,002 D: 0 PEIm1: 123137 PEIm2: 123137

B.3.6. 24 Hilos

— 5k — T: 0,748575 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,723584 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,715047 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,714507 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,668291 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,748338 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,684702 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,64615 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,659609 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,725765 D: 0 PEIm1: 7219 PEIm2: 7219 — 9k — T: 1,51605 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,48358 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,60885 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,65602 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,68141 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,7641 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,55316 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,57438 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,68858 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,02328 D: 0 PEIm1: 11198 PEIm2: 11198 — 11k — T: 3,0646 D: 0 PEIm1: 15040 PEIm2: 15040 T: 1,98462 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,02705 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,02464 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,97935 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,86123 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,01907 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,6761 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,94715 D: 0 PEIm1: 15040 PEIm2: 15040 T: 3,05071 D: 0 PEIm1: 15040 PEIm2: 15040 — 15k — T: 5,19854 D: 0 PEIm1: 20189 PEIm2: 20189 T: 5,26586 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,5453 D: 0 PEIm1: 20189 PEIm2: 20189 T: 5,43019 D: 0 PEIm1: 20189 PEIm2: 20189 T: 4,86951 D: 0 PEIm1: 20189 PEIm2: 20189 T: 5,49258 D: 0 PEIm1: 20189 PEIm2: 20189 T: 5,00268 D: 0 PEIm1: 20189 PEIm2: 20189 T: 5,75011 D: 0 PEIm1: 20189 PEIm2: 20189 T: 4,77999 D: 0 PEIm1: 20189 PEIm2: 20189 T: 5,43623 D: 0 PEIm1: 20189 PEIm2: 20189 — 19k — T: 3,09656 D: 0 PEIm1: 19490 PEIm2: 19490 T: 4,79578 D: 0 PEIm1: 19490 PEIm2: 19490 T: 4,46315 D: 0 PEIm1: 19490 PEIm2: 19490 T: 3,16372 D: 0 PEIm1: 19490 PEIm2: 19490 T: 3,09842 D: 0 PEIm1: 19490 PEIm2: 19490 T: 4,41453 D: 0 PEIm1: 19490 PEIm2: 19490 T: 5,07279 D: 0 PEIm1: 19490 PEIm2: 19490 T: 4,4658 D: 0 PEIm1: 19490 PEIm2: 19490 T: 4,55213 D: 0 PEIm1: 19490 PEIm2: 19490 T: 5,07089 D: 0 PEIm1: 19490 PEIm2: 19490 — 25k — T: 7,67931 D: 0 PEIm1: 25576 PEIm2: 25576 T: 7,71661 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,24569 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,24766 D: 0 PEIm1: 25576 PEIm2: 25576 T: 8,34486 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,22163 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,25254 D: 0 PEIm1: 25576 PEIm2: 25576 T: 7,90682 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,31158 D: 0 PEIm1: 25576 PEIm2: 25576 T: 7,6245 D: 0 PEIm1: 25576 PEIm2: 25576 — 27k — T: 6,24876 D: 0 PEIm1: 27927 PEIm2: 27927 T: 10,4509 D: 0 PEIm1: 27927 PEIm2: 27927 T: 8,69183 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,26032 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,25543 D: 0 PEIm1: 27927 PEIm2: 27927 T: 9,55202 D: 0 PEIm1: 27927 PEIm2: 27927 T: 9,77487 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,62581 D: 0 PEIm1: 27927 PEIm2: 27927 T: 10,349 D: 0 PEIm1: 27927 PEIm2: 27927 T: 9,92892 D: 0 PEIm1: 27927 PEIm2: 27927 — 30k — T: 11,8491 D: 0 PEIm1: 30337 PEIm2: 30337 T: 10,5819 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,54345 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,41913 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,42474 D: 0 PEIm1: 30337 PEIm2: 30337 T: 8,42991 D: 0 PEIm1: 30337 PEIm2: 30337 T: 11,4439 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,54241 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,41419 D: 0 PEIm1: 30337 PEIm2: 30337 T: 11,8802 D: 0 PEIm1: 30337 PEIm2: 30337 — 41k — T: 14,3444 D: 0 PEIm1: 41533 PEIm2: 41533 T: 20,7194 D: 0 PEIm1: 41533 PEIm2: 41533 T: 20,1486 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,7986 D: 0 PEIm1: 41533 PEIm2: 41533 T: 14,699 D: 0 PEIm1: 41533 PEIm2: 41533 T: 22,4883 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,9065 D: 0 PEIm1: 41533 PEIm2: 41533 T: 14,6887 D: 0 PEIm1: 41533 PEIm2: 41533 T: 22,1479 D: 0 PEIm1: 41533 PEIm2: 41533 T: 14,667 D: 0 PEIm1: 41533 PEIm2: 41533 — 47k — T: 21,0931 D: 0 PEIm1: 47141 PEIm2: 47141 T: 27,8964 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,8117 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,8137 D: 0 PEIm1: 47141 PEIm2: 47141 T: 19,174 D: 0 PEIm1: 47141 PEIm2: 47141 T: 30,729 D: 0 PEIm1: 47141 PEIm2: 47141 T: 29,5148 D: 0 PEIm1: 47141 PEIm2: 47141 T: 30,6612 D: 0 PEIm1: 47141 PEIm2: 47141 T: 29,7881 D: 0 PEIm1: 47141 PEIm2: 47141 T: 26,9138 D: 0 PEIm1: 47141 PEIm2: 47141 — 54k — T: 24,67 D: 0 PEIm1: 54895 PEIm2:

54895 T: 25,6746 D: 0 PEIm1: 54895 PEIm2: 54895 T: 24,7678 D: 0 PEIm1: 54895 PEIm2: 54895 T: 24,0909 D: 0 PEIm1: 54895 PEIm2: 54895 T: 35,5849 D: 0 PEIm1: 54895 PEIm2: 54895 T: 25,0736 D: 0 PEIm1: 54895 PEIm2: 54895 T: 37,731 D: 0 PEIm1: 54895 PEIm2: 54895 T: 24,5353 D: 0 PEIm1: 54895 PEIm2: 54895 T: 38,0514 D: 0 PEIm1: 54895 PEIm2: 54895 T: 33,5842 D: 0 PEIm1: 54895 PEIm2: 54895 — 57k — T: 33,4986 D: 0 PEIm1: 57726 PEIm2: 57726 T: 26,6311 D: 0 PEIm1: 57726 PEIm2: 57726 T: 28,1992 D: 0 PEIm1: 57726 PEIm2: 57726 T: 27,3937 D: 0 PEIm1: 57726 PEIm2: 57726 T: 27,4825 D: 0 PEIm1: 57726 PEIm2: 57726 T: 42,3663 D: 0 PEIm1: 57726 PEIm2: 57726 T: 26,6188 D: 0 PEIm1: 57726 PEIm2: 57726 T: 27,0325 D: 0 PEIm1: 57726 PEIm2: 57726 T: 26,7143 D: 0 PEIm1: 57726 PEIm2: 57726 T: 44,2161 D: 0 PEIm1: 57726 PEIm2: 57726 — 68k — T: 38,3138 D: 0 PEIm1: 68761 PEIm2: 68761 T: 60,1732 D: 0 PEIm1: 68761 PEIm2: 68761 T: 62,3841 D: 0 PEIm1: 68761 PEIm2: 68761 T: 55,6709 D: 0 PEIm1: 68761 PEIm2: 68761 T: 54,9025 D: 0 PEIm1: 68761 PEIm2: 68761 T: 40,6867 D: 0 PEIm1: 68761 PEIm2: 68761 T: 37,53 D: 0 PEIm1: 68761 PEIm2: 68761 T: 37,522 D: 0 PEIm1: 68761 PEIm2: 68761 T: 37,5525 D: 0 PEIm1: 68761 PEIm2: 68761 T: 38,3879 D: 0 PEIm1: 68761 PEIm2: 68761 — 84k — T: 56,1769 D: 0 PEIm1: 83942 PEIm2: 83942 T: 56,1561 D: 0 PEIm1: 83942 PEIm2: 83942 T: 82,0887 D: 0 PEIm1: 83942 PEIm2: 83942 T: 56,1298 D: 0 PEIm1: 83942 PEIm2: 83942 T: 56,1585 D: 0 PEIm1: 83942 PEIm2: 83942 T: 103,153 D: 0 PEIm1: 83942 PEIm2: 83942 T: 56,9397 D: 0 PEIm1: 83942 PEIm2: 83942 T: 82,7316 D: 0 PEIm1: 83942 PEIm2: 83942 T: 56,158 D: 0 PEIm1: 83942 PEIm2: 83942 T: 85,8398 D: 0 PEIm1: 83942 PEIm2: 83942 — 108k — T: 100,417 D: 0 PEIm1: 108076 PEIm2: 108076 T: 101,88 D: 0 PEIm1: 108076 PEIm2: 108076 T: 109,067 D: 0 PEIm1: 108076 PEIm2: 108076 T: 95,9597 D: 0 PEIm1: 108076 PEIm2: 108076 T: 94,3103 D: 0 PEIm1: 108076 PEIm2: 108076 T: 111,025 D: 0 PEIm1: 108076 PEIm2: 108076 T: 109,375 D: 0 PEIm1: 108076 PEIm2: 108076 T: 100,526 D: 0 PEIm1: 108076 PEIm2: 108076 T: 128,313 D: 0 PEIm1: 108076 PEIm2: 108076 T: 92,6921 D: 0 PEIm1: 108076 PEIm2: 108076 — 123k — T: 129,635 D: 0 PEIm1: 123137 PEIm2: 123137

B.3.7. 48 Hilos

— 5k — T: 0,729106 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,778212 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,757015 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,85927 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,771858 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,565566 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,790033 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,796414 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,793162 D: 0 PEIm1: 7219 PEIm2: 7219 T: 0,752866 D: 0 PEIm1: 7219 PEIm2: 7219 — 9k — T: 1,65896 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,22464 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,1598 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,15651 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,72663 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,60444 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,5981 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,514 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,15779 D: 0 PEIm1: 11198 PEIm2: 11198 T: 1,26519 D: 0 PEIm1: 11198 PEIm2: 11198 — 11k — T: 2,50291 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,64512 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,08958 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,5309 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,57762 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,47406 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,1196 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,0259 D: 0 PEIm1: 15040 PEIm2: 15040 T: 2,18907 D: 0 PEIm1: 15040 PEIm2: 15040 T: 1,84414 D: 0 PEIm1: 15040 PEIm2: 15040 — 15k — T: 3,60402 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,52043 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,80493 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,63927 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,89128 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,17529 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,51126 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,60583 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,16056 D: 0 PEIm1: 20189 PEIm2: 20189 T: 3,66308 D: 0 PEIm1: 20189 PEIm2: 20189 — 19k — T: 3,84205 D: 0 PEIm1: 19490 PEIm2: 19490 T: 2,91297 D: 0 PEIm1: 19490 PEIm2: 19490 T: 3,27394 D: 0 PEIm1: 19490 PEIm2: 19490 T: 4,14926 D: 0 PEIm1: 19490 PEIm2: 19490 T: 2,88674 D: 0 PEIm1: 19490 PEIm2: 19490 T: 2,91531 D: 0 PEIm1: 19490 PEIm2: 19490 T: 3,99805 D: 0 PEIm1: 19490 PEIm2: 19490 T: 3,13343 D: 0 PEIm1: 19490 PEIm2: 19490 T: 3,75313 D: 0 PEIm1: 19490 PEIm2: 19490 T: 3,27066 D: 0 PEIm1: 19490 PEIm2: 19490 — 25k — T: 5,3931 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,65887 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,22802 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,55483 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,4725 D: 0 PEIm1: 25576 PEIm2: 25576 T: 4,9959 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,83499 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,04681 D: 0 PEIm1: 25576 PEIm2: 25576 T: 5,27003 D: 0 PEIm1: 25576 PEIm2: 25576 T: 4,97962 D: 0 PEIm1: 25576 PEIm2: 25576 — 27k — T: 6,31651 D: 0 PEIm1: 27927 PEIm2: 27927 T: 5,91262 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,71092 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,14164 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,36632 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,10299 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,1619 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,42764 D: 0 PEIm1: 27927 PEIm2: 27927 T: 7,1481 D: 0 PEIm1: 27927 PEIm2: 27927 T: 6,51799 D: 0 PEIm1: 27927 PEIm2: 27927 — 30k — T: 7,17667 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,22255 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,6013 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,05754 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,08001 D: 0 PEIm1: 30337 PEIm2: 30337 T: 8,21794 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,9203 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,95289 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,69672 D: 0 PEIm1: 30337 PEIm2: 30337 T: 7,28869 D: 0 PEIm1: 30337 PEIm2: 30337 — 41k — T: 13,4196 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,3653 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,3283 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,1769 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,5905 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,1201 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,1494 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,8893 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,3707 D: 0 PEIm1: 41533 PEIm2: 41533 T: 13,5215 D: 0 PEIm1: 41533 PEIm2: 41533 — 47k — T: 17,8837 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,0957 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,1936 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,314 D: 0 PEIm1: 47141 PEIm2: 47141 T: 16,9943 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,1411 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,6523 D: 0 PEIm1: 47141 PEIm2: 47141 T: 16,9568 D: 0 PEIm1: 47141 PEIm2: 47141 T: 16,9211 D: 0 PEIm1: 47141 PEIm2: 47141 T: 17,7165 D: 0 PEIm1: 47141 PEIm2: 47141 — 54k — T: 23,2473 D: 0 PEIm1: 54895 PEIm2: 54895 T: 23,8021 D: 0 PEIm1: 54895 PEIm2: 54895 T: 22,5558 D: 0 PEIm1: 54895 PEIm2: 54895 T: 22,9396 D: 0 PEIm1: 54895 PEIm2: 54895 T: 22,7487 D: 0 PEIm1: 54895 PEIm2: 54895 T: 22,81 D: 0 PEIm1: 54895 PEIm2: 54895 T: 23,0204 D: 0 PEIm1: 54895 PEIm2: 54895 T: 22,9569 D: 0 PEIm1: 54895 PEIm2: 54895 T: 24,0225 D: 0 PEIm1: 54895 PEIm2: 54895 T: 23,104 D: 0 PEIm1: 54895 PEIm2: 54895 — 57k — T: 26,7299 D: 0 PEIm1: 57726 PEIm2: 57726 T: 26,3629 D: 0 PEIm1: 57726 PEIm2: 57726 T: 25,2192 D: 0 PEIm1: 57726 PEIm2: 57726 T: 24,9216 D: 0 PEIm1: 57726 PEIm2: 57726 T: 25,8032 D: 0 PEIm1: 57726 PEIm2: 57726 T: 25,3859 D: 0 PEIm1: 57726 PEIm2: 57726 T: 26,7924 D: 0 PEIm1: 57726 PEIm2: 57726 T: 25,0634 D: 0 PEIm1: 57726 PEIm2: 57726 T: 26,6253 D: 0 PEIm1: 57726 PEIm2: 57726 T: 25,6244 D: 0 PEIm1: 57726 PEIm2: 57726 — 68k — T: 35,5222 D: 0 PEIm1: 68761 PEIm2: 68761 T: 35,7351 D: 0 PEIm1: 68761 PEIm2: 68761 T: 36,6435 D: 0 PEIm1: 68761 PEIm2: 68761 T: 35,4358 D: 0 PEIm1: 68761 PEIm2: 68761 T: 35,5591 D: 0 PEIm1: 68761 PEIm2: 68761 T: 37,4432 D: 0 PEIm1: 68761 PEIm2: 68761 T: 35,251 D: 0 PEIm1: 68761 PEIm2: 68761 T: 35,9092 D: 0 PEIm1: 68761 PEIm2: 68761 T: 35,5831 D: 0 PEIm1: 68761 PEIm2: 68761 T: 35,9952 D: 0 PEIm1: 68761 PEIm2: 68761 — 84k — T: 55,2462 D: 0 PEIm1: 83942 PEIm2: 83942 T: 52,8156 D: 0 PEIm1: 83942 PEIm2: 83942 T: 53,1942 D: 0 PEIm1: 83942 PEIm2: 83942 T: 52,7972 D: 0 PEIm1: 83942 PEIm2: 83942 T: 52,8856 D: 0 PEIm1: 83942 PEIm2: 83942 T: 56,101 D: 0 PEIm1: 83942 PEIm2: 83942 T: 53,5964 D: 0 PEIm1: 83942 PEIm2: 83942 T: 52,6099 D: 0 PEIm1: 83942 PEIm2: 83942 T: 52,9218 D: 0 PEIm1: 83942 PEIm2: 83942 T: 56,0093 D: 0 PEIm1: 83942 PEIm2: 83942 — 108k — T: 88,2697 D: 0 PEIm1: 108076 PEIm2: 108076 T: 87,3039 D: 0 PEIm1: 108076 PEIm2: 108076 T: 92,4884 D: 0 PEIm1: 108076 PEIm2: 108076 T: 87,6553 D: 0 PEIm1: 108076 PEIm2: 108076 T: 87,2338 D: 0 PEIm1: 108076 PEIm2: 108076 T: 92,6399 D: 0 PEIm1: 108076 PEIm2: 108076 T: 87,2448 D: 0 PEIm1: 108076 PEIm2: 108076 T: 87,5419 D: 0 PEIm1: 108076 PEIm2: 108076 T: 87,7464 D: 0 PEIm1: 108076 PEIm2: 108076 T: 87,6172 D: 0 PEIm1: 108076 PEIm2: 108076 — 123k — T: 113,428 D: 0 PEIm1: 123137 PEIm2: 123137 T: 114,37 D: 0 PEIm1: 123137 PEIm2: 123137 T: 119,296 D: 0 PEIm1: 123137 PEIm2: 123137 T: 119,655 D: 0 PEIm1: 123137 PEIm2: 123137 T: 112,942 D: 0 PEIm1: 123137 PEIm2: 123137

Agradecimientos

Agradezco a mis padres todo el apoyo y paciencia que me han dado a lo largo de los años para poder llegar hasta donde estoy ahora. A mi compañera Ana por soportar mis nervios y ansiedad a lo largo de los años de carrera y animarme cuando me encontraba bajo de fuerzas. Y a mis compañeros de prácticas por las horas de esfuerzo conjunto que nos han permitido llegar hasta aqui.

Bibliografía

A. Ashbrook, N. T. 1998. *Tutorial: Algorithms For 2-Dimensional Object Recognition.* University of Manchester,Stopford Building, Oxford Road, Manchester, M13 9PT.

AMD. 2011. Opencl and the amd sdk.

Amdahl, G. M. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference.* AFIPS '67 (Spring). ACM, New York, NY, USA, 483–485.

Anwar Ghuloum, Amanda Sharp, N. C. 2010. Array building blocks: A flexible parallel programming model for multicore and many-core architectures. webinar.

Barney, B. 1998. *Tutorial: OpenMP.* Lawrence Livermore National Laboratory.

for Computer Science, S. T. G. M. L. 1998. Cilk 5.4.6 reference manual. webpage.

Group, K. O. W. 2011. *The OpenCL Specification.*

Hausdorff, F. 1914. *Grundzuge der Mengenlehre.* January 1, 1949 Ed. American Mathematical Society, 416 Fourth Street, Ann Arbor, MI 48103-4820 USA.

Huttenlocher, D., Klanderman, G., and Rucklidge, W. 1993. Comparing images using the hausdorff distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 15,* 9, 850 –863.

Intel. 2009. Intel® cilk++ sdk programmer's guide. webpage.

Intel. 2012. Intel threading building blocks for open source documentation. webpage.

nVidia. 2012a. *CUDA C Programming Guide.* nVidia.

nVidia. 2012b. *CUDA Toolkit Reference Manual.* nVidia.

Sanchez, L., Fernandez, J., Sotomayor, R., and Garcia, J. 2012. A comparative evaluation of parallel programming models for shared-memory architectures. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on.* 363 –370.

Shameem Akhter, J. R. 2006. *Multi-Core Programming: Increasing Performance through Software Multi-threading.* Intel Press.

Suau, P. 2005. Adapting hausdorff metrics to face detection systems: A scale-normalized hausdorff distance approach. In *Progress in Artificial Intelligence*, C. Bento, A. Cardoso, and G. Dias, Eds. Lecture Notes in Computer Science Series, vol. 3808. Springer Berlin / Heidelberg, 76–86.

Williams, A. 2011. *C++ Concurrency in Action: Practical Multithreading.* Manning Pubs Co Series. Manning Publications.